Program Synthesis Meets Deep Learning for Decoding Regulatory Networks

Jasmin Fisher [a,b*] and Steven Woodhouse[b]

[a] Department of Biochemistry, University of Cambridge, Cambridge CB2 1QW, UK;
[b] Microsoft Research, Cambridge CB1 2FB, UK

[*] Correspondence should be addressed to Jasmin Fisher at jf416@cam.ac.uk
Department of Biochemistry
University of Cambridge
Hopkins Building, Downing Site
Cambridge, CB2 1QW, UK
Phone: +44-1223 479 947
Fax: +44-1223 479 999

Abstract

With ever growing data sets spanning DNA sequencing all the way to single-cell transcriptomics, we are now facing the question of how can we turn this vast amount of information into knowledge. How do we integrate these large data sets into a coherent whole to help understand biological programs? The last few years have seen a growing interest in machine learning methods to analyse patterns in high-throughput data sets and an increasing interest in using program synthesis techniques to reconstruct and analyse executable models of gene regulatory networks. In this review, we discuss the synergies between the two methods and share our views on how they can be combined to reconstruct executable mechanistic programs directly from large-scale genomic data.

Introduction

Uncovering and understanding the programs that underlie the behaviour of cells is one of the major challenges in biology today. Understanding these programs will help us determine the molecular mechanisms of disease, and ultimately impact translational research. A central goal of executable biology [1] is the construction of executable mechanistic models of such cellular behaviour programs, and the development of computational techniques for automated analysis and inference of these models. In this review, we discuss two fields of computer science, machine learning and program synthesis that are focused on learning predictive models from data and on automated construction of computer programs from desired behaviours, respectively. These two fields can be seen as two sides of the same coin, particularly in the context of executable biology where we want to learn from large complex datasets, and where the artefact we ideally want to learn is a mechanistic model of the cell's behaviour, which is essentially a program.

Recent research has begun to blur the boundaries between these two fields, as machine learning researchers have begun to develop methods that can learn algorithmic patterns in data and as programming language researchers have begun to investigate the methods of deep learning for program synthesis. Here, we explain the differences between these two approaches, discuss the growing connections between the two fields, and give our projections for how they can be combined to extract comprehensive cell signalling programs from the tsunami of genomic data.

Machine learning for uncovering patterns in large data sets

Machine learning allows us to approach problems that have no clear solution as a traditional program authored in human-readable source code [2], [3]. For example, how would you program a computer to recognise images of cats? There is no direct way to do this. Instead, one can use a machine learning algorithm to train a model on many images of cats, and have the algorithm learn the underlying patterns in the data in a way that generalises to new, previously unseen images. Then, when presented with a new image, the trained model is able to correctly predict whether it contains a cat or not.

Machine learning can be used for classification problems (for example, object recognition – detect a face in an image), for regression (i.e., predict a continuous variable given some input), or for sample generation (i.e., generate new objects that are similar to previously seen objects). A machine learning algorithm takes training data as an input and uses it to estimate a function f. The algorithm typically does this by optimising a metric that measures how well f fits the training data. Machine learning is distinguished from a regular optimisation problem by the requirement for generalisation to new, previously unseen data (in order not to over fit the training data). To test how well the trained model generalises, we evaluate the performance measure on test data, which is disjoint from the training data.

Classical approaches to machine learning

'Classical' machine learning approaches are based upon the careful extraction of *features* of a data set. These features are then plugged into a standard regression or classification algorithm, such as linear regression, naïve bayes, or the k-nearest neighbours classifier (see [3] for an

overview of these classic algorithms). The success or failure of the machine learning algorithm to make accurate predictions is largely dependent on the features it is presented with. In areas such as speech recognition and image classification, features of interest can be highly complex and must be thoroughly designed by hand by a domain expert, a process known as *feature engineering*.

An example of using classical machine learning techniques in biology is the prediction of gene expression levels from transcription factor binding profiles using linear regression [4]. Yu *et al.* map genotype to phenotype in yeast using random forests with gene ontology terms as features [5]. Deng *et al.* built a predictive system for classifying genes as essential or non-essential in bacteria by integrating gene expression, protein-protein interaction and genomic data, averaging the predictions of an ensemble of different models [6]. Applications of machine learning techniques to gene regulatory network reconstruction have focused on detection of statistical signals in gene expression data using clustering [7]–[9], correlation [10], [11], mutual information [12], Bayesian networks [13] or random forests [14].

Deep neural networks

Deep learning, the application of deep neural networks to machine learning, is the current state-of-the-art in supervised learning [2]. While the explosion of deep learning research is recent, researchers have been working on the underlying models, artificial neural networks, since the 1940s, initially as computational models of the brain. Early applications of deep learning in biology include the use of neural networks to decipher the complex tissue-specific splicing regulatory code and to predict DNA-protein binding [15], [16].

There are two major features that distinguish deep learning from classical approaches to machine learning. The first is that neural networks can represent essentially any (continuous) function, rather than simple functions of a specific form [17]. This property is true of shallow neural networks, as well as deep ones. The second major difference between deep learning and classical machine learning is that deep neural networks perform *representation learning*. Representation learning solves the feature engineering problem faced by classical approaches to machine learning, mentioned above. In a deep neural network, the features themselves can also be learnt from the raw data, automatically. Figure 1 shows how a deep neural network can learn to represent the concept of an image of a person by building a hierarchy of representations, of increasing levels of abstraction [2].

Figure 2 shows an illustration of a deep neural network architecture for predicting the sequence specificities of DNA- and RNA-binding proteins [15]. The neural network is trained on sequence specificities measured by a range of experimental methodologies. The network then learns to generalise the patterns it finds in this data to discover both motifs and an associated score predicting their binding affinity. The resulting trained model can then be used to identify new binding sequences or predict the effect of DNA or RNA mutations.

In a neural network, components called neurons are connected into a graph. A neuron receives a signal from each of its input neighbours, takes a weighted combination of these inputs (according to learned weights) and passes the result through an activation function to determine its output signal. In a deep neural network, these outputs are in turn fed into other neurons, and many layers can be stacked, with the input to the system arriving at the first layer and the output of the system arriving from the final layer.

To train a neural network, we define an *objective function* that measures how well the network outputs fit our training data. In modern machine learning, the objective function and the activation functions are differentiable, meaning that a small change in the weights of a neuron result in small changes to its output. We can track the effect of changing each neuron weight on the resulting objective value, and use a local optimisation algorithm to iteratively update all the weights to optimize the objective. Because the objective function associated with a neural network is in general highly non-convex, a local optimisation algorithm may get stuck at a locally optimal, but not globally optimal choice of weights. It is therefore somewhat surprising that deep learning works so well. For image and speech recognition tasks, local optima which are far from the value of the global optimum do not seem to be a problem in practice. There has been some recent theoretical work on trying to understand this [18]–[21]. However, the presence of local optima seems to be a much larger issue for using deep learning approaches to synthesise programs, as we discuss next.

Program synthesis for reconstructing gene regulatory networks

Program synthesis is a method for automatically constructing a program that satisfies a given set of desired behaviours [22]–[25]. The set of behaviours can be given as a logical formula or as a set of input-output examples that the program should reproduce, or as some combination of the two. For example, we may want a program that can sort a list of integers. Rather than directly writing such a program, we may ask the computer to automatically find one for us. Unlike deep learning, program synthesis generally leads to discrete problems which can be exactly solved to obtain a globally optimal solution, using algorithms that leverage SAT, SMT, or integer linear programming solvers.

The Single Cell Network Synthesis toolkit (SCNS) is a method for synthesising executable models of gene regulatory networks in the form of Boolean networks from single-cell gene expression data [26], [27]. SCNS is based upon viewing single-cell gene expression profiles as though they were states of an asynchronous Boolean network, and then solving the problem of reconstructing a Boolean network from its state space. This algorithm uses a combination of enumerative search, graph reachability and Boolean satisfiability solving to extract a gene regulatory network model that best matches the state space data (Figure 3a). Before SCNS can be used, gene expression data first must be discretised to binary data, where continuous gene expression values are converted to Boolean on/off values.

The SCNS approach can be applied to study developmental processes, and requires measurement of sufficient single-cells to get reasonable coverage of a system across a time course. We applied this methodology to study early blood development in the mouse embryo, capturing nearly 4000 cells with blood-forming potential across four sequential time points. We designed this experiment so that approximately one embryo equivalent of cells was collected at each time point, giving a comprehensive single-cell resolution picture of the developmental process and allowing us to find a model that can explain transitions from early cell states to late cell states. Once a model has been found, it can be used to make predictions which can be validated experimentally. If the model predicts that transcription factors A and B are both required for activation of C, experiments can be designed that mutate binding sites for A and B individually and in combination and assess the effects on the expression of C. If

the model predicts that overexpression or knockout of a specific gene eliminates or adds model states, this can be tested via corresponding overexpression/knockout studies.

The Reasoning Engine for Interaction Networks (RE:IN) synthesises Boolean networks from prior knowledge of the gene regulatory network connections together with a set of desired stable states that the constructed model should have [28] (Figure 3b). The search for a compatible model is encoded as a logical formula and solved using an SMT solver. Again, this specification is discrete, given by binary stable states and the presence of network edges. Dunn *et al*. used this method to reconstruct a minimal Boolean network model that can explain embryonic stem cell pluripotency.

Connections between machine learning and program synthesis for learning programs from data

Recently, machine learning researchers have begun to extend their deep neural network models so that they can learn algorithmic patterns in data. At the same time, researchers working in program synthesis have begun to investigate the methods of deep learning for program synthesis, and so these two fields have begun to overlap. Below we survey the latest developments in these fields, and in the next section we will discuss how these advances could be used to improve methods for synthesising biological models.

Deep learning researchers have found that by augmenting networks with an external data structure such as a tape, stack or list, they can train models to learn simple algorithms [29]–[38]. Compared to regular programs, there is no interpretable source code representation for these trained models. They are black boxes given by a huge number of parameters on a neural network, and they can only be understood by their actions on given inputs [39].

On the other side, in the programming languages community, there has been work on applying the methods of differentiable models and deep learning to the problem of synthesising (the source code of) programs. The tool TerpreT has been developed to understand the capabilities of machine learning techniques relative to traditional alternatives based on discrete models and exact constraint solving [39]. The conclusions of this study were that constraint solving significantly outperformed the direct application of machine learning methods. Gaunt *et al.* also showed that on a simple problem there are exponentially many local optima in the solution space and that empirically they arise often in practice during training. However, the use of differentiable models and deep learning does provide some interesting possibilities, such as synthesising programs from perceptual data such as images [40]. Differentiable Forth is a similar work, where a Forth program with "holes" is sketched by the user, and then the holes are filled in by machine learning methods [41].

In Adaptive Neural Compilation, Bunel *et al.* introduce an approach for compiling an existing program to a differentiable model, then try to find a more efficient model for solving the same problem, by demanding that it agrees with the original program on a set of input-output examples [42]. The resulting model is then translated back to source code. In Neuro-Symbolic Program Synthesis, Parisotto *et al.* use a novel neural network architecture to search over the space of source code to find a matching program [43]. Very recently, Balog *et al.* introduced DeepCoder, an approach which uses neural networks to guide traditional search techniques

rather than directly using machine learning methods to search through the space of programs [44]. This approach combines the advantages of program synthesis and machine learning to scale to harder problems.

Challenges and future directions

From the viewpoint of executable biology, the task of automatically generating mechanistic insights from genomic data amounts to reconstructing gene regulatory network programs. This task poses two main challenges: (1) scalability – due to the sheer volume of data available, and (2) handling the non-discrete and often noisy nature of that data. Combining program synthesis with deep learning can potentially help address both of these problems (Figure 4).

S**calability**. While the potential for uncovering biological mechanisms from large data is huge, combinatorial synthesis methods such as those used in the Single Cell Network Synthesis tool – SCNS [26], [27] and the RE:IN tool [28] do not scale up sufficiently. Recently, combinatorial search methods augmented by a machine learning component to guide the search process have been successfully applied to improve baseline methods for automated theorem proving [45], [46] and program synthesis [44] and there is hope for further progress.

N**on-discrete data**. Current combinatorial synthesis methods rely on clean, discrete data and well-defined behaviours. Some valuable data sources, however, such as single-cell mass cytometry [47], are noisy and hard to discretise. A major attraction of machine learning methods is their ability to deal with noisy, continuous data. Thus, a combination of program synthesis and machine learning methods could reconstruct executable models directly from continuous expression data. Functions describing gene regulatory interactions could be represented as compact regularised neural networks. Then, using techniques developed in recent research on differential interpreters [39], [42], [44], [48]–[50], these networks could be translated back to a human-readable formula and a logical, executable model. There is now a deluge of data from DNA sequencing [51], imaging, proteomics [47] and metabolomics [52] that as of yet has not been leveraged to reconstruct mechanistic models. The potential for uncovering biological mechanisms from this data is huge, but will require both the scalability and robustness to noise found in machine learning methods and the ability to extract executable, human-interpretable models found in methods developed in the program synthesis community.

Going forward, combining deep learning and program synthesis will allow us to incorporate into executable models information from diverse data sources, which may be continuous, noisy or perceptual and therefore difficult to deal with using existing methods. Taken together, these developments hold the promise of innovative methods for turning genomic datasets into a comprehensive map of human cells in health and disease.

Acknowledgements

The authors thank Nir Piterman, Marc Brockschmidt, Alexander Gaunt, Daniel Tarlow and Christoph Wintersteiger for valuable discussions. This work was supported in part by Microsoft Research.

Figures

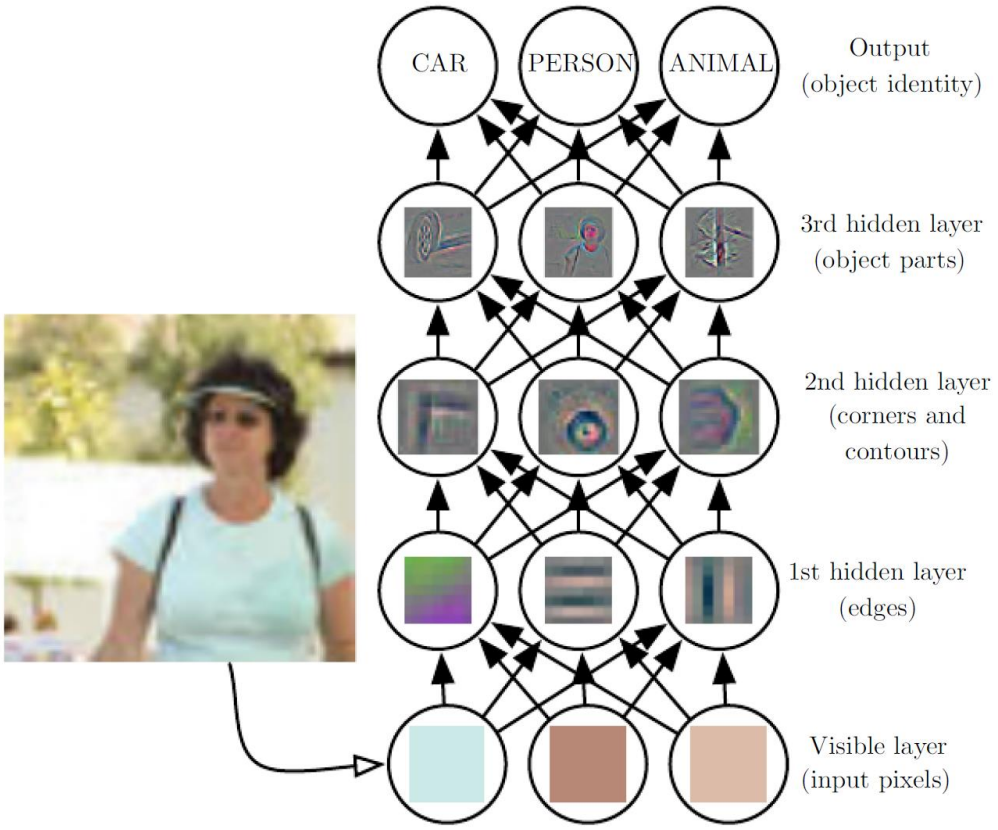

Figure 1. Representation learning. Deep neural networks learn to represent concepts in terms of progressively simpler ones. At the lowest level of the network, raw pixel values are input into the model. The next layer of the model identifies edges, by comparing the brightness of neighbouring pixels. The third layer takes the representation of edges, and uses them to represent corners and contours. The forth layer is able to detect entire parts of specific objects, by combing together contours and corners. Finally, the model outputs a classification of the image which it determines based upon the object parts fed from the forth layer. Crucially, none of these abstract concepts is provided *a-priori* by the programmer. Instead, they are directly learnt from the raw data. Reproduced, with permission, from [2].

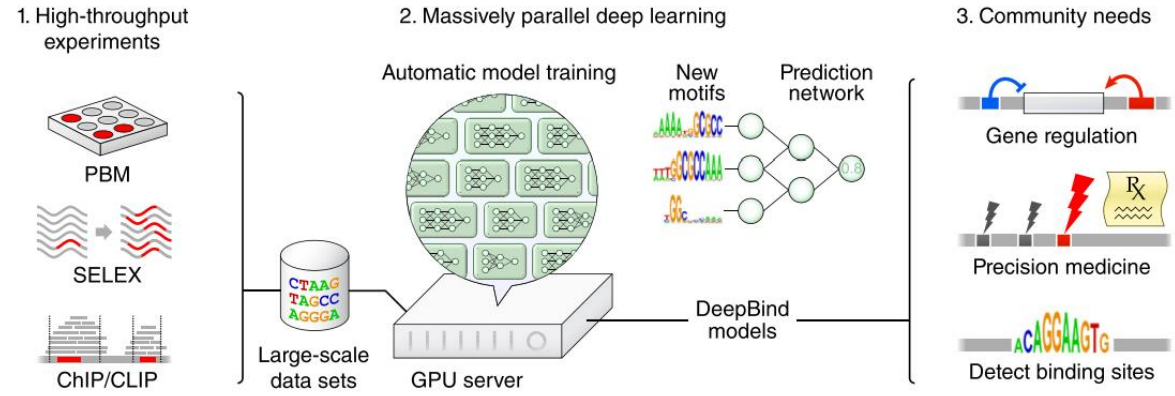

Figure 2. Deep learning in biology. A deep neural network architecture for predicting the sequence specificities of DNA- and RNA-binding proteins. Reproduced, with permission, from [15].

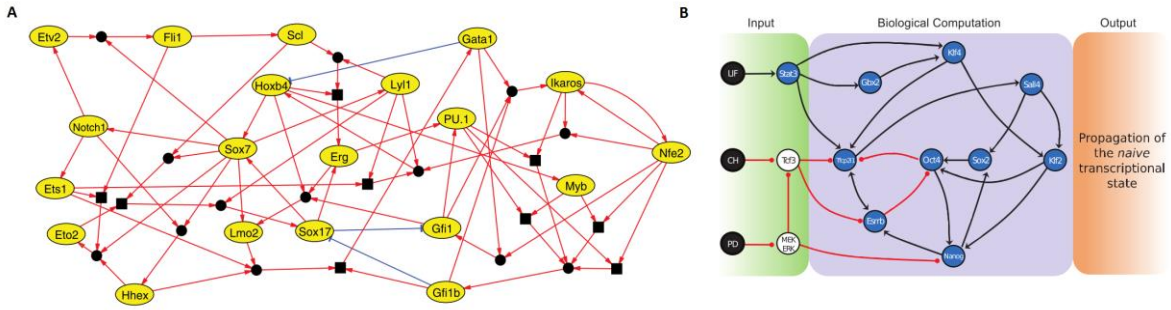

Figure 3. Executable gene regulatory network model synthesis. Boolean network models synthesised by SCNS (A) and REIN (B), from single-cell gene expression data and from relevance networks + stable state specifications respectively. Reproduced, with permission, from [27] and [28].

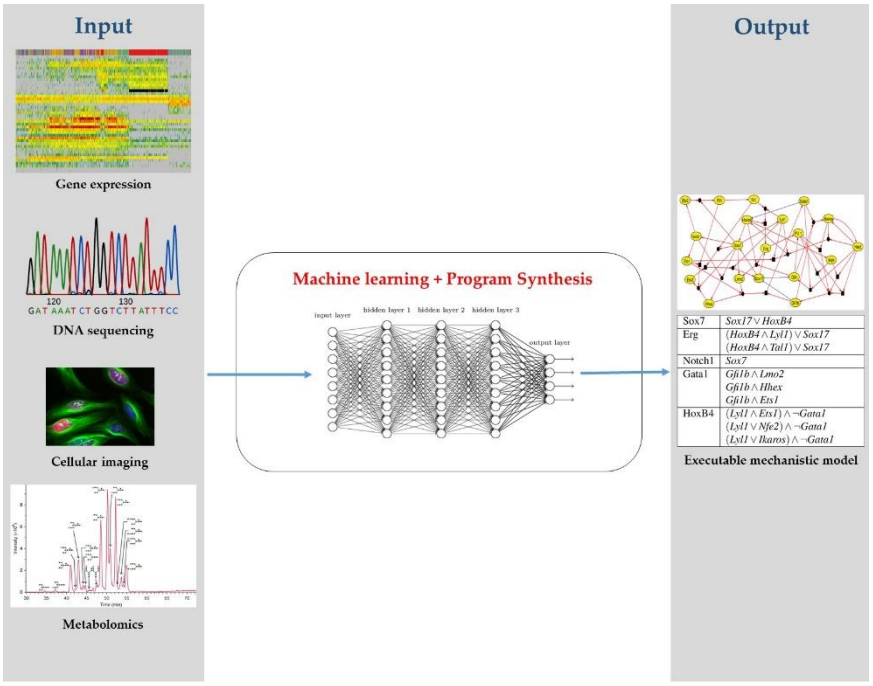

Figure 4. Combining representation learning and program synthesis to reconstruct human-interpretable, executable models from a variety of biological data sources. In our first proposed approach, using machine learning to scale synthesis to larger data sets, a neural network will output a probability distribution over features of the data. This will be used to guide a constraint solver, by looking for models with high-probability features first. In the second proposed approach, neural network methods will be directly applied to learn a model from non-discretised data, which will be translated to a human-interpretable form.

References

Of outstanding interest:

[2] Very up-to-date and comprehensive introduction to modern deep neural networks.
[37] Differentiable neural computers are introduced, a type of model that can access and manipulate an external data source to perform algorithmic tasks, and learn to do so from data.
[39] Combinatorial and deep learning methods are rigorously compared on program synthesis tasks.
[44] Demonstrates how deep learning and program synthesis can be combined to increase scalability on a program synthesis problem. A neural network guides a combinatorial solver by pointing it in the direction of more likely solutions.

Of special interest:

[27] Program synthesis techniques are used to derive an executable gene regulatory network for early blood development, using single-cell gene expression data.
[28] Program synthesis techniques are used to derive an executable gene regulatory network for embryonic stem cell pluripotency.
[42] Demonstrates how programs can be compiled to differentiable representations, optimised to make them more efficent using deep learning methods, and then can be translated back to human-interpretable source code.

[1] J. Fisher and T. A. Henzinger, "Executable Cell Biology," *Nat. Biotechnol.*, vol. 25, no. 11, pp. 1239–1249, 2007.
[2] A. Goodfellow, Ian, Bengio, Yoshua, Courville, "Deep Learning," *MIT Press*, 2016. [Online]. Available: http://www.deeplearningbook.org/.
[3] C. M. Bishop, *Pattern Recognition and Machine Learning*, vol. 4, no. 4. 2006.
[4] Z. Ouyang, Q. Zhou, and W. H. Wong, "ChIP-Seq of transcription factors predicts absolute and differential gene expression in embryonic stem cells," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 106, no. 51, pp. 21521–21526, 2009.
[5] M. K. Yu *et al.*, "Translation of genotype to phenotype by a hierarchy of cell subsystems," *Cell Syst.*, vol. 2, no. 2, pp. 77–88, 2016.
[6] J. Deng *et al.*, "Investigating the predictability of essential genes across distantly related organisms using an integrative approach," *Nucleic Acids Res.*, vol. 39, no. 3, pp. 795–807, 2011.
[7] P. D'haeseleer, "How does gene expression clustering work?," *Nat. Biotechnol.*, vol. 23, no. 12, pp. 1499–1501, 2005.
[8] M. Hofree, J. P. Shen, H. Carter, A. Gross, and T. Ideker, "Network-based stratification of tumor mutations," *Nat. Methods*, vol. 10, no. 11, pp. 1108–1115, 2013.
[9] P. Creixell *et al.*, "Pathway and network analysis of cancer genomes," *Nat. Methods*, vol. 12, no. 7, pp. 615–621, 2015.
[10] V. Moignard *et al.*, "Characterization of transcriptional networks in blood stem and progenitor cells using high-throughput single-cell gene expression analysis.," *Nat. Cell*

Biol., vol. 15, no. 4, pp. 363–372, 2013.

[11] P. A. Jaeger *et al.*, "Network-driven plasma proteomics expose molecular changes in the Alzheimer's brain," *Mol. Neurodegener.*, vol. 11, no. 1, p. 31, 2016.

[12] S. Krishnaswamy *et al.*, "Conditional density-based analysis of T cell signaling in single-cell data."

[13] K. Sachs, O. Perez, D. Pe'er, D. a Lauffenburger, and G. P. Nolan, "Causal protein-signaling networks derived from multiparameter single-cell data.," *Science*, vol. 308, no. 5721, pp. 523–529, 2005.

[14] V. A. Huynh-Thu, A. Irrthum, L. Wehenkel, and P. Geurts, "Inferring regulatory networks from expression data using tree-based methods," *PLoS One*, vol. 5, no. 9, pp. 1–10, 2010.

[15] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey, "Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning.," *Nat Biotechnol*, vol. 33, no. 8, pp. 831–838, 2015.

[16] M. K. K. Leung, H. Y. Xiong, L. J. Lee, and B. J. Frey, "Deep learning of the tissue-regulated splicing code," *Bioinformatics*, vol. 30, no. 12, 2014.

[17] G. Cybenko, "Approximations by superpositions of sigmoidal functions," *Approx. Theory its Appl.*, vol. 9, no. 3, pp. 17–28, 1989.

[18] A. Choromanska, M. Henaff, M. Mathieu, G. Ben Arous, and Y. LeCun, "The Loss Surfaces of Multilayer Networks," *Aistats*, vol. 38, pp. 192--204, 2015.

[19] Y. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," *arXiv*, pp. 1–14, 2014.

[20] L. Sagun, V. U. Guney, and Y. LeCun, "Explorations on high dimensional landscapes," *Iclr 2015*, no. 2012, pp. 1–10, 2014.

[21] K. Kawaguchi, "Deep Learning without Poor Local Minima," *NIPS*, no. Nips, 2016.

[22] M. Vardi, "From Verification to Synthesis," *Verif. Softw. Theor. Tools, Exp.*, p. 2, 2008.

[23] S. Srivastava, S. Gulwani, and J. S. Foster, "From program verification to program synthesis," *ACM SIGPLAN Not.*, vol. 45, p. 313, 2010.

[24] K. Ellis, A. Solar-lezama, and J. B. Tenenbaum, "Unsupervised Learning by Program Synthesis," *Nips*, pp. 1–9, 2015.

[25] A. Solar-Lezama, R. M. Rabbah, R. Bodík, and K. Ebcioglu, "Programming by sketching for bit-streaming programs," in *Programming Language Design and Implementation*, 2005, pp. 281–294.

[26] J. Fisher, A. S. Köksal, N. Piterman, and S. Woodhouse, "Synthesising Executable Gene Regulatory Networks from Single-cell Gene Expression Data," *Comput. Aided Verif. (CAV), Springer*, 2015.

[27] V. Moignard *et al.*, "Decoding the regulatory network of early blood development from single-cell gene expression measurements," *Nat. Biotechnol.*, vol. 33, no. 3, pp. 269–76, 2015.

[28] S.-J. Dunn, G. Martello, B. Yordanov, S. Emmott, and a. G. Smith, "Defining an essential transcription factor program for naive pluripotency," *Science (80-.).*, vol. 344, no. 6188, pp. 1156–1160, 2014.

[29] Ł. Kaiser and I. Sutskever, "Neural GPUs Learn Algorithms," *arXiv1511.08228 [cs]*, pp. 1–9, 2015.

[30] A. Graves, G. Wayne, and I. Danihelka, "Neural Turing Machines," *Arxiv*, pp. 1–26, 2014.

[31] A. Joulin and T. Mikolov, "Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets," *arXiv*, pp. 1–10, 2015.
[32] W. Zaremba, T. Mikolov, A. Joulin, and R. Fergus, "Learning Simple Algorithms from Examples," *arXiv*, pp. 1–12, 2015.
[33] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-To-End Memory Networks," *Adv. Neural Inf. Process. Syst. (NIPS '15)*, pp. 1–11, 2015.
[34] E. Grefenstette, K. M. Hermann, M. Suleyman, and P. Blunsom, "Learning to Transduce with Unbounded Memory," *arXiv*, p. 12, 2015.
[35] S. Reed and N. de Freitas, "Neural Programmer-Interpreters," in *ICLR*, 2016, pp. 1–13.
[36] K. Kurach, M. Andrychowicz, and I. Sutskever, "Neural Random-Access Machines," in *International Conference On Learning Representations*, 2016, pp. 1–13.
[37] A. Graves *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471–476, Oct. 2016.
[38] W. I. T. H. Spss, W. Evel, and H. L. M. A. Nalysis, "Making Neural Programming Architectures Generalize via Recursion," in *ICLR*, 2017, no. 2016, pp. 0–3.
[39] A. L. Gaunt *et al.*, "TerpreT: A Probabilistic Programming Language for Program Induction," *arXiv*, 2016.
[40] A. L. Gaunt, M. Brockschmidt, N. Kushman, and D. Tarlow, "Differentiable Programs with Neural Libraries," *arXiv*, Nov. 2016.
[41] S. Riedel, M. Bošnjak, and T. Rocktäschel, "Programming with a Differentiable Forth Interpreter," *Arxiv*, 2016.
[42] R. Bunel, A. Desmaison, P. Kohli, P. H. S. Torr, and M. Pawan Kumar, "Adaptive Neural Compilation," *arXiv:1605.07969*, 2016.
[43] E. Parisotto, A. Mohamed, R. Singh, L. Li, D. Zhou, and P. Kohli, "Neuro-Symbolic Program Synthesis," Nov. 2016.
[44] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, "DeepCoder: Learning to Write Programs," Nov. 2016.
[45] G. Irving, C. Szegedy, A. A. Alemi, N. Een, F. Chollet, and J. Urban, "DeepMath - Deep Sequence Models for Premise Selection," in *NIPS*, 2016, pp. 2235–2243.
[46] S. Loos, G. Irving, C. Szegedy, and C. Kaliszyk, "Deep Network Guided Proof Search," Jan. 2017.
[47] E. R. Zunder, E. Lujan, Y. Goltsev, M. Wernig, and G. P. Nolan, "A continuous molecular roadmap to iPSC reprogramming through progression analysis of single-cell mass cytometry.," *Cell Stem Cell*, vol. 16, no. 3, pp. 323–37, 2015.
[48] S. Reed and N. de Freitas, "Neural Programmer-Interpreters," *ICLR*, pp. 1–13, 2016.
[49] T. Rocktaschel, S. Riedel, T. Rockt, and S. Riedel, "Learning Knowledge Base Inference with Neural Theorem Provers," *Proc. 5th Work. Autom. Knowl. Base Constr.*, pp. 45–50, 2016.
[50] A. L. Gaunt, M. Brockschmidt, N. Kushman, and D. Tarlow, "Lifelong Perceptual Programming By Example," Nov. 2016.
[51] R. Gao *et al.*, "Punctuated copy number evolution and clonal stasis in triple-negative breast cancer," *Nat. Genet.*, no. August, pp. 1–15, 2016.
[52] C. H. Johnson, J. Ivanisevic, and G. Siuzdak, "Metabolomics: beyond biomarkers and towards mechanisms.," *Nat. Rev. Mol. Cell Biol.*, vol. 17, no. 7, pp. 451–9, 2016.