RESEARCH

Differential Architecture Search in Deep Learning for DNA Splice Site Classification

Shabir Moosa[1,2*†]
, Prof. Abbes Amira[2]
and Dr. Sabri Boughorbel[1†]

*Correspondence:
smoosa@sidra.org
[1]Department of Systems Biology, SIDRA Medicine, 26999 Doha, Qatar
Full list of author information is available at the end of the article
[†]Equal contributor

Abstract

Background: The data explosion caused by unprecedented advancements in the field of genomics is constantly challenging the conventional methods used in the interpretation of the human genome. The demand for robust algorithms over the recent years has brought huge success in the field of Deep Learning (DL) in solving many difficult tasks in image, speech and natural language processing by automating the manual process of architecture design. This has been fueled through the development of new DL architectures. Yet genomics possesses unique challenges as we expect DL to provide a super human intelligence that easily interprets a human genome.

Methods: We adapted a differential architecture search method for the interpretation of biological sequences and applied it to the splice site recognition task on DNA sequences to discover new high-performance convolutional architectures in an automated manner. The discovered architecture was benchmarked on CPU and multiple GPU architectures in terms of computational time and classification performance.

Results: Our experimental evaluation demonstrated that the discovered architecture outperformed fixed baseline architectures for classification of splice sites. The benchmarking experiments of execution time and precision on architecture search and evaluation process showed that they performed better on recently available GPU models.

Conclusions: We applied differential architecture search mechanism to perform splice site classification on raw DNA sequences, and discovered new models with better performance than major baseline models. The results have shown a potential of using this automated architecture search mechanism for solving various problems in genomics domain.

Keywords: Deep Learning; Splice Site; Genomics; Neural Architecture Search; Convolutional Neural Networks

Background

Deep Learning is a class of Machine Learning (ML) algorithms that combines raw inputs into layers of intermediate features. They take raw features from large datasets and use them to create a predictive tool from hidden patterns in the data. They have shown impressive results over existing best-in class ML algorithms across various domains. For the past five years, DL algorithms have revolutionized fields such as high-energy physics [1], computational chemistry [2], dermatology [3]. The off-the-shelf implementation of these algorithms across different fields have produced

comparable or higher accuracies than previous state-of-the art methods that required extensive customization over the years.

The advancement of neural networks have demonstrated revolutionizing achievements in the field of image classification, object detection and natural language processing. Designing these neural network architectures requires computational resources and significant efforts from human experts in DL through trial and error. Over the recent years, there has been a paradigm shift from feature designing to architecture designing in the field of image classification and natural language processing [4, 5, 6, 7] to develop algorithmic solutions for automating the manual process of architecture design using Neural Architecture Search (NAS) methods. They have provided promising results in designing models better than human designed ones on benchmark datasets. The goal in architecture search is to find an optimal architecture from a given search space so that the validation accuracy on a particular task is maximized. NAS has some similarities to program synthesis and inductive programming where a program is searched from examples [8]. Many architecture search algorithms have been proposed such as Reinforcement Learning (RL) [7] which uses a policy gradient algorithm to optimize architecture configurations. This approach is computationally expensive and time consuming as they design and train each network from scratch during the exploration in the search space. Several approaches were proposed for improving the efficiency of NAS such as establishing a particular structure for the search space [4], performance prediction [9] and weight prediction of individual architectures [10] and by a parameter sharing mechanism [11] across multiple architectures. A novel approach of searching the architectures over a continuous domain alternate to searching over a discrete set of child architectures was proposed in [12]. This Differential Architecture Search (DARTS) mechanism has outperformed various other architecture search algorithms by achieving competitive performance over a rich architecture search space by using less computation resources.

In this paper, the DARTS technique used in [12] for image classification tasks was adapted to solve problems in genomics domain. The study was performed on the Splice Site Recognition (SSR) dataset which provides a scope for analysis of the human genome and identification of unknown regions to understand the biochemical processes involved in building and maintaining a human body.

Splice Site Recognition Problem

Proteins form an essential component in all living organism and a major biological process that occurs in all living cells is the production of proteins. They play a vital role in the biochemical reactions within cells and in metabolism. The protein coding process called gene expression occurs in two stages: Transcription and Translation. During Transcription , the DNA (Deoxyribonucleic Acid) is synthesized to produce an mRNA (messenger Ribonucleic Acid). The protein coding process occurs in the translation phase where the mRNA sequence is decoded to produce the proteins.

Prokaryotic organisms do not have a cell nucleus and their translation stage is relatively simpler. But in eukaryotic organisms the genes are composed of alternated segments of introns and exons. Exons forms the coding regions in a DNA during translation to proteins. The biological significance of intronic regions are not known

yet as they do not participate in the protein building process. During translation stage in eukaryotes, the process of splicing occurs where the introns are spliced out from the mRNA molecule. The boundary points where splicing occurs on a gene sequence are called splice junction sites or splice sites as shown in Fig. 1.

Precise identification of Exon-Intron (EI) junctions or donors and Intron-Exon junctions (IE) or acceptors from a sequence is beneficial for advancements in transcriptome research and is a crucial step for fully understanding the gene expression. The accurate detection of splice junctions is challenging because of the high rate of false positives caused by the presence of short canonical splicing patterns [13]. There are currently two different techniques used to solve the splice junction prediction problem: Alignment based techniques and ML based techniques. The sequence alignment-based techniques maps millions of short RNA sequences produced by RNA-seq to the reference genome and then estimate where splicing occur by identifying the adjacent exon locations. The existing alignment based techniques such as SpliceMap [14] and TopHat [15] detects only canonical splice sites while missing the non-canonical sites which are required for accurate junction prediction. The ML based techniques can predict non-canonical sites as well by appropriate training. Different ML approaches has been used such as Support Vector Machines (SVM) [16, 17, 18, 19] Random Forest (RF) [20], Decision Trees (DT) [21], Naïve Bayesian (NB) [22], Markov Model [23] and AdaBoost [24] to identify splice or non-splice sites. Among them SVM models have been used very often due to their capability to handle high-dimensional datasets. However, certain kernel and penalty parameters in SVM require extensive tuning which is time consuming. The effectiveness of all these approaches also depends on the feature engineering technique used which is a major initial step in solving a classification problem. Many feature engineering techniques have been proposed for feature construction directly from the DNA sequence ,such as the MM1 (1-order Markov model) in [16] for feature construction from splice site sequences and using the SVM for prediction. In [20] and [17] features were constructed based different statistical approach with automated feature extraction was proposed in [22] for prediction of splice sites. A length-variable Markov Model (LVMM) which produced higher accuracy with low time cost was discussed by [23]. A hybrid algorithm of AdaBoost classifier was proposed in [24] which provided an improvement in performance compared to the other approaches. The efficacy of all these approaches is based on the feature extraction step which is often a tedious task that is performed by domain experts. Manual operation of feature extraction often leads to incomplete representation or one-high dimensional feature space which will cause problems in the machine learning process. The challenges involved in performing manual feature extraction and model training has led to a demand for DL based computational methods that performed automated feature representation. Many DL architectures were used and developed for the splice site prediction based on CNN [25, 26], RNN [13, 27], Restricted Boltzmann Machines (RBM) [28], Autoencoders [29, 30] and Deep Belief Networks [28]. Although these DL architectures have removed the burden of manual feature extraction, they are still time consuming to train and a much deeper knowledge on splice sites-associated functions and evolution has been strongly urged. In general, the existing methods still undergoes the manual effort in designing architectures which needs a lot of expert domain knowledge and is time consuming.

Methods

Our model makes use of deep CNN to distinguish features between true and false splice junctions. CNN architectures have shown better performance in learning features that classify actual splice sites from false ones. The method consists of two stages: Architecture Search and Architecture Evaluation.In the first stage, architecture search using DARTS was performed to discover the best model and the second stage validates the discovered model on a held out unseen data. The model gains from the information present in the genomic sequence of the candidate splice junction to accurately classify whether the sequence corresponds to a splice junction or not.

Dataset

The experiments were performed using Splice Site Recognition (SSR) dataset that is publicly available at [31]. The underlying problem posed in this dataset is to classify, given a sequence of DNA, as a splice or a non-splice sequence. The splice junctions are locations in a DNA sequence where 'superfluous' DNA is removed during protein creation process. The beginning and end of an exon is determined by the splice-donor and splice-acceptor sequences present. In this study, the prediction of splice junctions are performed using the given annotated DNA sequences with true acceptor splice site sequences. The original dataset has 159,771 true acceptor splice site sequences and 14,868,555 non-acceptor sequences. To avoid the class imbalance problem we used all the true acceptor splice site sequences and equal number of non-acceptor sequences chosen randomly. The dataset used in our study has a total of 319542 DNA sequence samples with a sequence length of 141 base pair. The dataset was randomly split into three sets using customized data loaders into train, validation and test datasets respectively of size 0.6, 0.2, 0.2.

DNA Representation

Genome sequence data is biologically described using four types of nucleotide, adenine (A), cytosine (C), guanine(G) and thymine (T). Each of these sequences are converted into numerical representation using 1-dimensional orthogonal encoding or one-hot encoding for downstream analysis. However, to shape the input appropriately for the DARTS Convolutional Neural Networks (CNN) model, the DNA sequences are represented as a 3-dimensional tensor, which is similar in shape to an image tensor input to image classification networks. Firstly, one-hot encoding is applied which converts each nucleotide in the DNA sequence of length n_d into a four-dimensional vector and then concatenates each of them to form the complete sequence. The next step is to transform the list of one-hot vectors to a 3-dimensional tensor.

let s ϵ S where S ={A,T,C,G}, then, a sequence (A,C,G,T,A,C) will be encoded into a tuple of 4-D binary vectors.

([1,0,0,0],[0,0,1,0],[0,0,0,1],[0,1,0,0],[1,0,0,0],[0,0,1,0])

The encoded sequence is then represented as a three-dimensional matrix of shape (3 x n_d x 4). The final representation of the input to the model will be in the form (batch size x 3 x n_d x 4)

DARTS Algorithm

The DARTS model discovers state-of-the art network architectures by formulating the task in a differential manner. The interesting part in this method is that the search space is treated as continuous rather than searching over a discrete set of architectures in the search space.

The cell in the architecture is considered as a Direct Acyclic Graph (DAG) consisting of a set of nodes and edges. Each cell has one output node and two input nodes. Let N be the number of nodes and each node represented by x^i . Each edge (i, j) performs an operation represented by $o^{(i,j)}$ that transforms x^i. The intermediate nodes are computed based on its predecessors.

$$x^i = \sum_{j<i} (o^{i,j} x^j) \tag{1}$$

The learning of the cell involves learning the operations that transform the input. Fig. 2 shows an overview of the DARTS method [12]. The goal of the method is to find a cell that forms the building block of the final architecture. Initially the operations on the edges are unknown. Let O be the set of operations where each operation is represented as $o(.)$ to be applied to x^i. The choice of the operation is made in a continuous manner by performing a softmax on all possible operations.

$$o^{-(i,j)}(x) = \sum_{o \epsilon O} \frac{\exp(\alpha_0^{(i,j)})}{\sum_{o' \epsilon O} \exp(\alpha_{0'}^{()})} o(x) \tag{2}$$

Here $\alpha_0^{(i,j)}$ is a vector with dimension $|O|$ that indicates the mixing of operation between a pair of nodes. The architecture search phase jointly performs learning on a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$ and the weights(ω) within each operation in O. The value of α and ω is obtained through a bi-level optimization algorithm where α becomes the higher level variable and ω acts as the lower level variable. The search finds a value of α that minimizes the validation loss L_{val} for that value of ω that minimizes the training loss L_{train}.

$$\min_{\alpha} \qquad L_{valid}(\omega^*(\alpha), \alpha) \tag{3}$$

$$s.t \quad \omega^*(\alpha) = argmin_{\omega} \quad L_{train}(\omega, \alpha) \tag{4}$$

This bi-level optimization algorithm 1 shows the optimization of ω and α in the respective search spaces through a gradient-based approach. The operation at each edge is replaced by the operation that had the maximum value of α.

Algorithm 1: Optimization algorithm of ω w.r.t α

Let $o^{-(i,j)}$ be the set of operations parameterized by $\alpha^{(i,j)}$ for an edge $pair(i, j)$
while *not converged* **do**
    compute ω by decreasing $\nabla_{\omega} L_{train}(\omega, \alpha)$;
    compute α by decreasing $\nabla_{\alpha} L_{valid}(\omega - \xi \nabla_{\omega} L_{train}(\omega, \alpha), \alpha)$;
Replace $o^{-(i,j)}$ with $o^{(i,j)} = argmax_{o \epsilon O}\ \alpha_o^{i,j}$ for each pair of edge(i, j)

The optimization is performed in the architecture search algorithm 2 during the training phase and the best discrete architecture $arch_{final}$ is saved to be evaluated on the architecture evaluation algorithm 3. The training phase in the architecture evaluation phase has a fixed architecture $arch_{final}$ and is then trained to obtain the optimal architecture weights. The trained model is evaluated on unseen data in the architecture evaluation phase.

Algorithm 2: Architecture Search Algorithm

Phase1: Architecture Search
Input : x_s=($x_{s,1}$,$x_{s,2}$,$x_{s,3}$,...,$x_{s,|x_s|}$) $x_s \epsilon X_s$
where X: a set of sequences with $|X|$=N
x_s: a single sequence with length $|x_s|$
$x_{(s,i)} \epsilon \{A, T, C, G\}$ for $i = 1, 2, 3....|x_s|$
y: label for x_s. $y \epsilon \{0, 1\}$
where 0 means non-splice and 1 represents splice sequence
Output: $arch_{final}$
where $arch_{final}$ is the final architecture cell

```
/* Data Preprocessing and Loading                                    */
```
$X_T \leftarrow transform(X_s)$; where X_T: Transformed dataset
$(x_t, x_v) \leftarrow split(X_T)$; where x_t: Training set and x_v: Validation set
initialize best accuracy acc_b ;
for *each epoch* **do**
```
    /* Training                                                      */
```
    for *m,n training data randomly selected from* x_t,x_v **do**
        initialize new architecture $arch(\omega, \alpha)$ using Algorithm 1;
        train($arch(\omega, \alpha)$); ; `// Training the model`
```
    /* Validation                                                    */
```
    for *p validation data randomly selected from* x_v **do**
        $acc_{val} = valid(arch(\omega, \alpha))$; ; `// Validating the model`
    if $acc_{val} > acc_b$ **then**
        acc_b=acc_{val};
        $arch_{final} = arch(\omega, \alpha)$;
save $arch_{final}$ for architecture evaluation phase;

Experimental Setup and Implementation

The architecture search and evaluation experiments were performed on Sidra HPC environment using NVIDIA Tesla K40M. The implementation was done using Pytorch which is an open-source library in python based on Torch that supports strong GPU acceleration.

The computational benchmarking experiments were performed on Intel(R) Xeon(R) x86_64, Quadro K4100M and Tesla V100-SXM2 architectures. The specification of the hardware architectures used is listed in Table 1. The GPUs and CPUs were configured with the same environment as the original experiments. The CPU evaluation was performed by submitting the task as a job to the cluster environment.

Architecture Search

The following operations were only included from a rich primitive space used in [12] for our search space O :3x3 separable convolutions, 3x3 dilated separable convolutions, 3x3 max pooling, 3x3 average pooling and zero. A building block of a convolution operation is formed by three steps: Firstly an activation function is Rectifier Linear Unit (RELU) applied and then a convolution operation (CN) is executed and finally, a batch normalization (BN) is performed. This is denoted as

Algorithm 3: Architecture Evaluation Algorithm

Phase2: Architecture Evaluation
Input : x_s=($x_{s,1}$,$x_{s,2}$,$x_{s,3}$,...,$x_{s,|x_s|}$) $x_s \epsilon X_s$
where X: a set of sequences with $|X|$=N
x_s: a single sequence with length $|x_s|$
$x_{(s,i)} \epsilon \{A, T, C, G\}$ for $i = 1, 2, 3....|x_s|$
y: label for x_s. $y \epsilon \{0, 1\}$
where 0 means non-splice and 1 represents splice sequence
Output: acc_{final}
where acc_{final} is the final accuracy
/* Data Preprocessing and Loading */
$X_T \leftarrow transform(X_s)$; where X_T: Transformed dataset
$(x_t, x_v, x_f) \leftarrow split(X_T)$; where x_t: Training set, x_v: Validation set and x_f: Test set
initialize best accuracy acc_b ;
initialize weights ω;
initialize architecture $arch$ to $arch_{final}$ obtained from Algorithm 2;
for *each epoch* **do**
  /* Training */
  for m *training data randomly selected from* x_t **do**
    train($arch(\omega, \alpha)$); ; // Training the model
    update weights ω using SGD with momentum ;
  /* Validation */
  for p *validation data randomly selected from* x_v **do**
    $acc_{val} = valid(arch(\omega, \alpha))$; ; // Validating the trained model
  if $acc_{val} > acc_b$ **then**
    acc_b=acc_{val};
    save best model $arch(\omega, \alpha)$;
acc_{final}=test($arch(\omega, \alpha)$); ; // Testing the final model

ReLU-Conv-BN and we used the same ReLU-Conv-BN order in [12] for performing convolutional operations. Our discovered convolutional cell consisted of 4 nodes, where the output node is result of the depth-wise concatenation of the convolution and pooling layers excluding the input node. The final architecture network was formed by stacking multiple cells together. The architecture consists of two types of convolutional cells called normal cell and reduction cell to make it scalable for any input size. When a feature map is taken as input, the normal cell returns a feature map of same dimension. The reduction cell returns a feature map where the height and width are reduced by a factor of two. The reduction cells are located at 1/3 and 2/3 of the total depth of the architecture. The architecture has a reduction cell in every third cell of the complete architecture. The first and second input nodes of the cell k are set to the $k-2$ and $k-1$ cells respectively. The architecture search was performed on train and validation datasets. A network composed of 8 cells were trained for 50 epochs using DARTS with batch size 500 set for both training and validation. The weights ω were optimized using Stochastic Gradient Descent (SGD) with momentum and Adam as the optimizer for architecture variables. The initial learning rate was set as 0.0025 and was gradually decreased to a minimum of 0.001. The rest of the hyperparameters were chosen similar to the original implementation in [12] as shown in Table 2.

Architecture Evaluation

The best architecture cells shown in Fig. 3 were selected based on the performance on validation dataset. In order to select the optimal architecture for evaluation, the search experiment was run ten times with different seeds. The best performing cell was recorded during epoch 29 in the ninth iteration. A network of 3 cells were trained

for 40 epochs with batch size 500. The rest of the hyperparameters were similar to the ones used for the architecture search process. The selected best architecture was evaluated using a held out test dataset. It is important to note that the test set was never used during the architecture search process. As the results were subject to variance even for same configurations, the mean and standard deviation of 10 independent runs was reported.

Results

Classification Performance Results

The splice dataset classification results using various DL techniques are presented in Table 3. The average and standard deviation metrics were calculated over 10 repetitions of each experiment. The DARTS experiment was repeated by running the experiments on 10 different seeds. The experiments on the baseline models were repeated by k-fold cross-validation. The performance of the automatically discovered architecture was compared against fixed baseline CNN and Recurrent Neural Network (RNN) models. The baseline models were implemented using keras framework to benchmark against known models from the literature. Notably, DARTS outperformed over the widely known Long Short Term Memory (LSTM) and baseline CNN with and without embedding in terms of test accuracy, sensitivity, specificity, F-score and AUC score. The model achieved better accuracy results over the hybrid and LSTM model with embedding.

Fig. 4 shows the mean and standard deviation plots of training and Fig. 5 shows the validation metrics over 40 epochs for all the baseline models against our proposed model. Notably, the fixed CNN baseline architectures performed poorly compared to other architectures.

Computational Performance Results

In addition, the proposed model was benchmarked on GPU systems and CPU for comparison of execution time and precision as shown in Fig. 6

Execution Time

The execution time for performing architecture search and architecture evaluation were calculated on different device architectures and the results are presented in Table 4. Notably, the most advanced Tesla V100 GPU completed the search in less than 11 hours and the evaluation in half an hour.

Precision

The learning and inference speed of the trained model were compared on different GPU architectures as shown in Table 5. The experiments were performed on single precision, half precision, double precision data types. The model was fed with a single batch input of 500 sequences. For training, the time required for 20 forward and backward passes were averaged. In inference, time duration of 20 forward passes were averaged. Five warm up steps were included that was not calculated towards the final results.

Discussion

Deep Learning is an emerging research topic among the genomics community. Its applications can be revolutionized by introducing high-performance computing methods to analyze datasets in the field of gene therapies, molecular diagnostics and personalized medicine. In the scope of this paper, an advanced DL approach based on differential architecture search was implemented to solve the splice site classification problem in genomics and to discover new high performance CNN architectures. It was observed in the literature review that most of the work was focused on using manually designed architectures in ML and DL. This study has aided in bridging the gap between the state-of-the art in DL and its application to genomics. The evaluation results showed that the newly discovered architecture outperformed the fixed baseline DL architectures using the same dataset. The architecture was compared alongside the well-known LSTM model and complex hybrid architectures. Furthermore, the discovered architecture was evaluated on multiple CPU and GPU architectures. The total time taken for performing the architecture search and evaluation were determined as well as the floating point instructions per second for single, double and half precision were compared. The computational benchmarking results obtained proved that there is significant improvement in execution time when using advanced GPU architectures.

For all its promises, DL in genomics still possess a number of challenges. The results largely depend on the quality of the data input that are well annotated so that the model can learn to distinguish features and identify patterns. Another challenge is the lack of judgement capability where the technique is able to distinguish from a biologically relevant variation and normal variations. This would require applying further experimental design and controls. The advancements in the field of DL in the field of computer vision and speech recognition has led to new methods being constantly proposed that awaits its application in genomics domain. Furthermore, the availability of quasi-unlimited storage at a reasonable price, the surge in computing power and the lower computational costs will allow these advanced DL techniques to reshape the capabilities of machines to completely understand and interpret the human genome.

Conclusions

In this study, we applied the differential architecture search technique for performing classification on raw DNA sequences and compared the newly discovered architecture against well known fixed baseline architectures. As future steps, the plan is to further improve the performance of DARTS based on CNN by including more primitives such as skip connect and higher convolutional operations, thereby widening the architecture search space. This will help to traverse more information to lower layers. The DARTS approach will further be evaluated against the recent parallel work of Neural Architecture Optimization (NAO) which also performs continuous optimization of architecture space. The study showed that fixed RNN architectures have better results than CNN. Genomics data are sequential data similar to speech recognition, natural language processing and language translation. It would be interesting to implement DARTS to search for a recurrent cell that can be recursively connected to form a RNN that can be applied for tasks of protein function prediction. As part of performance benchmarking, the plan is to evaluate the models

further on GPU clusters and Google's Tensor Processing Unit (TPU) which are specialized hardware architectures developed to accelerate AI workloads. In addition, this approach will be tested on future genomics classification tasks, as it will be highly useful to uncover new insights from the vast available sequencing data.

Abbreviations
CNN: Convolutional Neural Networks; DARTS: Differential Architecture Search; DL: Deep Learning; DNA: Deoxyribonucleic Acid; LSTM: Long Short Term Memory; ML: Machine Learning; NAS: Neural Architecture Search; NAO: Neural Architecture Optimization; RNN: Recurrent Neural Network; SGD: Stochastic Gradient Descent

Acknowledgements
We would like to thank the HPC team at SIDRA for their computational support with the experimental analysis.

Author's contributions
SM implemented the method, conducted the experiments, and drafted the manuscript. SB and AA supervised every step of the work and provided critical review and valuable input. All authors read and approved the final manuscript.

Funding
Not applicable

Availability of data and materials
The data used in the experiments is publicly available at https://bmi.inf.ethz.ch/supplements/lsmkl.

Ethics approval and consent to participate
Not applicable

Consent for publication
Not applicable

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Department of Systems Biology, SIDRA Medicine, 26999 Doha, Qatar. [2]Dept. of Computer Science and Engineering, Qatar University, 2713 Doha, Qatar.

References
1. Baldi, P., Sadowski, P., Whiteson, D.: Searching for exotic particles in high-energy physics with deep learning. Nature communications **5**, 4308 (2014)
2. Goh, G.B., Hodas, N.O., Vishnu, A.: Deep learning for computational chemistry. Journal of computational chemistry **38**(16), 1291–1307 (2017)
3. Esteva, A., Kuprel, B., Novoa, R.A., Ko, J., Swetter, S.M., Blau, H.M., Thrun, S.: Dermatologist-level classification of skin cancer with deep neural networks. Nature **542**(7639), 115 (2017)
4. Liu, H., Simonyan, K., Vinyals, O., Fernando, C., Kavukcuoglu, K.: Hierarchical representations for efficient architecture search. arXiv preprint arXiv:1711.00436 (2017)
5. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. arXiv preprint arXiv:1802.01548 (2018)
6. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)
7. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. arXiv preprint arXiv:1707.07012 **2**(6) (2017)
8. Summers, P.D.: A methodology for lisp program construction from examples. Journal of the ACM (JACM) **24**(1), 161–175 (1977)
9. Baker, B., Gupta, O., Raskar, R., Naik, N.: Accelerating neural architecture search using performance prediction. arXiv preprint arXiv:1705.10823 (2017)
10. Brock, A., Lim, T., Ritchie, J.M., Weston, N.: Smash: one-shot model architecture search through hypernetworks. arXiv preprint arXiv:1708.05344 (2017)
11. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268 (2018)
12. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search. arXiv preprint arXiv:1806.09055 (2018)
13. Lee, B., Lee, T., Na, B., Yoon, S.: Dna-level splice junction prediction using deep recurrent neural networks. arXiv preprint arXiv:1512.05135 (2015)
14. Au, K.F., Jiang, H., Lin, L., Xing, Y., Wong, W.H.: Detection of splice junctions from paired-end rna-seq data by splicemap. Nucleic acids research **38**(14), 4570–4578 (2010)
15. Trapnell, C., Pachter, L., Salzberg, S.L.: Tophat: discovering splice junctions with rna-seq. Bioinformatics **25**(9), 1105–1111 (2009)
16. Baten, A.K., Chang, B.C., Halgamuge, S.K., Li, J.: Splice site identification using probabilistic parameters and svm classification. In: BMC Bioinformatics, vol. 7, p. 15 (2006). BioMed Central

17. Meher, P.K., Sahu, T.K., Rao, A., Wahi, S.: Identification of donor splice sites using support vector machine: a computational approach based on positional, compositional and dependency features. Algorithms for molecular biology **11**(1), 16 (2016)
18. Zhang, Y., Chu, C.-H., Chen, Y., Zha, H., Ji, X.: Splice site prediction using support vector machines with a bayes kernel. Expert Systems with Applications **30**(1), 73–81 (2006)
19. Wei, D., Zhuang, W., Jiang, Q., Wei, Y.: A new classification method for human gene splice site prediction. In: International Conference on Health Information Science, pp. 121–130 (2012). Springer
20. Meher, P.K., Sahu, T.K., Rao, A.R.: Prediction of donor splice sites using random forest with a new sequence encoding approach. BioData mining **9**(1), 4 (2016)
21. Lopes, H.S., Erig Lima, C.R., Murata, N.J.: A configware approach for high-speed parallel analysis of genomic data. Journal of Circuits, Systems, and Computers **16**(04), 527–540 (2007)
22. Kamath, U., De Jong, K., Shehu, A.: Effective automated feature construction and selection for classification of biological sequences. PloS one **9**(7), 99982 (2014)
23. Zhang, Q., Peng, Q., Zhang, Q., Yan, Y., Li, K., Li, J.: Splice sites prediction of human genome using length-variable markov model and feature selection. Expert Systems with Applications **37**(4), 2771–2782 (2010)
24. Pashaei, E., Yilmaz, A., Ozen, M., Aydin, N.: Prediction of splice site using adaboost with a new sequence encoding approach. In: Systems, Man, and Cybernetics (SMC), 2016 IEEE International Conference On, pp. 003853–003858 (2016). IEEE
25. Elsousy, R., Kathiresan, N., Boughorbel, S.: On the depth of deep learning models for splice site identification. bioRxiv, 380667 (2018)
26. Du, X., Yao, Y., Diao, Y., Zhu, H., Zhang, Y., Li, S.: Deepss: Exploring splice site motif through convolutional neural network directly from dna sequence. IEEE Access **6**, 32958–32978 (2018)
27. Kothen-Hill, S.T., Zviran, A., Schulman, R.C., Deochand, S., Gaiti, F., Maloney, D., Huang, K.Y., Liao, W., Robine, N., Omans, N.D., et al.: Deep learning mutation prediction enables early stage lung cancer detection in liquid biopsy (2018)
28. Lee, T., Yoon, S.: Boosted categorical restricted boltzmann machine for computational prediction of splice junctions. In: International Conference on Machine Learning, pp. 2483–2492 (2015)
29. Lee, B., Baek, J., Park, S., Yoon, S.: deeptarget: end-to-end learning framework for microrna target prediction using deep recurrent neural networks. In: Proceedings of the 7th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, pp. 434–442 (2016). ACM
30. Xu, Z.-C., Wang, P., Qiu, W.-R., Xiao, X.: iss-pc: Identifying splicing sites via physical-chemical properties using deep sparse auto-encoder. Scientific Reports **7**(1), 8222 (2017)
31. Sonnenburg, S., Rätsch, G., Schäfer, C., Schölkopf, B.: Large scale multiple kernel learning. Journal of Machine Learning Research **7**(Jul), 1531–1565 (2006)

Figures

Figure 1 Splice-junction sites on a sequence where the splicing occurs at the exon-intron pairs

Figure 2 An overview of DARTS method where the architecture cell is discovered by formulating the search task of the operations between each node in a differential manner.

Figure 3 Normal and Reduction cell learned on splice dataset.The normal cell (panel **a**) returns feature maps with the same dimension whereas dimensions of the feature maps of reduction cell (panel **b**) is reduced by a factor of two.

Figure 4 Training accuracy (panel **a**) and loss (panel **b**) measurements of the discovered model against fixed architectures.

Figure 5 Validation accuracy (panel **a**) and loss (panel **b**) measurements of the discovered model against fixed architectures.

Tables

Figure 6 Search and evaluation execution time of the DARTS model on GPU and CPU devices (panel **a**) and training and inference speed of DARTS model on each of the devices (panel **b**).

Table 1 Hardware for Benchmarking

Device Model	Number of available cores	Memory size(in GB)
Tesla V100-SXM2	5120	16
Tesla K40m	2880	12
Quadro K4100M	1152	4
Intel(R) Xeon(R) CPU E5-2670	32	256

Table 2 Hyperparameters for architecture search

Hyperparameter	Value
Batch size	500
Initial learning rate	0.0003
Architecture learning rate	0.0025
Minimum learning rate.	0.001
Epochs	50
Weight decay rate	0.0003
Loss function	Cross Entropy
Update strategy	SGD with momentum

Table 3 Comparison of Model Performance

Model	Accuracy	Sensitivity	Specificity	F-score	AUC score
DARTS	**94.15 ± 0.1**	94.00 ± 0.1	95.20± 0.3	94 ± 0.1	**94.8 ± 0.01**
LSTM (emb.)	93.98 ± 0.1	95 ± 1.0	92.32 ± 1.2	93.74 ± 0.2	93.66 ± 0.2
Hybrid	93.66 ± 0.2	94.63 ± 0.7	93.33 ± 0.6	94.02 ± 0.1	93.66 ± 0.2
LSTM	86.99 ± 14.6	85.26 ± 19.8	88.71 ± 9.8	85.87 ± 16.9	86.99 ± 14.6
CNN	64.15 ± 0.6	65.27 ± 6.0	63.04 ± 6.2	64.43 ± 2.1	64.10 ± 0.5
CNN (emb.)	53.63 ± 3.6	95.62 ± 6.6	11.63 ± 13.7	67.35 ± 0.5	53.63 ± 3.6

Table 4 Comparison of Execution Time

Device Model	Architecture Search(hours)	Architecture Evaluation(hours)
Tesla V100-SXM2	10.75	0.5
Tesla K40m	38.5	2
Quadro K4100M	101	50
Intel(R) Xeon(R) CPU E5-2670	526	74

Table 5 Comparison of Learning and Inference Speed on GPUs

GPU Model	Training			Inference		
	Single	Half	Double	Single	Half	Double
Quadro K4100M	1041.24	1090.38	1902.38	134.97	124.59	274.78
Tesla K40m	352.53	470.72	485.17	48.022	49.38	62.13
Tesla V100-SXM2	68.46	119.55	130.09	13.21	11.02	11.07