Journal of Animal Science and Products (JASP)
International peer-reviewed journal
4 (2): 225-239, 2021

Some Deep Learning Algorithms Using in Complex Traits Genomic Prediction

Hayrettin OKUT

University of Kansas, School of Medicine, USA

Hayrettin OKUT, ORCID No: 0000-0003-4084-8404

ARTICLE INFO

Review

Received : 22.12.2021
Accepted : 30.12.2021

Keywords

Deep learning
Complex traits
Genomic prediction

*** Corresponding Author**

hokut@kumc.edu

ABSTRACT

The underlying perception of genomic selection (GS) is to use genome-wide from DNA sequence ("SNP markers") along with phenotypes from an observed population to make prediction for the phenotypic outcomes of untested individuals in crop and livestock breeding programs. GS was firstly described by Meuwissen et al. (2001) in dairy cattle to identify genetically superior animals at an early age. The aim was to capture specific genes across the whole genome that are associated with desired traits. The major challenge in using GS programs is to predict the effect of many SNP markers using phenotypic information from a few individuals (aka small n big p problem, or p >> n). Many approaches including naïve and scaled elastic net, ridge regression BLUP Bayesian approaches (BayesA, BayesB, BayesCπ, BayesDπ) LASSO, Support Vector Regression have been conducted to address the small n big p (aka, p >> n) problem. These methods all perform well for (p>>n) by using linear approximation to set a functional relationship between genotypes and phenotypes. However, these methods may not fully capture non-linear effects which are possible to be crucial for complex traits. To deal with this limitation, many methods including neural networks (NN) were recommended to cover non-linearity for GS. Artificial NNs (ANNs) for GS was first presented by Okut et al. (2011) who establish a fully connected regularized multi-layer ANN (MLANN) comprising one hidden layer to predict the body mass index (BMI) in mice using dense molecular markers. Since then, rather complex ANNs approaches have been applied including deep learning (DL) networks. The different DL algorithms have their own advantages to deal with specific problems in complex trait GS. Four different major classes of DL approaches such as fully connected deep learning artificial neural networks (DL-MLANN), recurrent neural networks (RNN), convolutional neural networks (CNN) and long-short term memory (LSTM) and some variation of these network architectures will be summarized here.

Kompleks Özelliklerde Geneomik Seleksiyon için Kullanılan Derin Öğrenim Algoritmaları

MAKALE BİLGİSİ

Derleme

Geliş: 22.12.2021
Kabul: 30.12.2021

ÖZ

Genomik seleksiyon (GS), bitki ve hayvan popülasyonundan gözlenemiyen fenotip ve DNA (SNP belirtiçleri) bilgisi kullanılarak ileriye yönelik fenotipik değerlerinin tahmin edilmesi amaçlanmaktadır. GS ilk olarak süt sığırcılığında erken yaşlarda genetiksel olaraka üstün bireylerin belirlenmesi amaçlanmıştır. Meuwissen ve arkadaşları tarafından 2001 yılında yürütülen bu çalışmada bütün genom içerisinde bazı önemli özellikler ile ilişkili genlerin ortaya

Lütfen aşağıdaki şekilde atıf yapınız / Please cite this paper as following;
Okut, H., 2021. Some deep learning algorithms using in complex traits genomic prediction, Journal of Animal Science and Products (JASP), 4 (2): 225-239. DOI: 10.51970/jasp.1039713

Anahtar Kelimeler

Derin öğrenme
Complex fenotipik özellikler
Genomic tahminleme

*** Sorumlu Yazar**

hokut@kumc.edu

koyulmasına çalışılmıştır. GS seleksiyon çalışmalarında bazı zorluklar söz konusudur. En önemli sorun, sadece çok az miktardaki bireye ait fenotipik değer kullanılarak çok miktardaki SNP belirteçin etkisisni araştırmaktır. Teknik anlamada bu soun küçük n büyük p (p>>n) olarak isimlendirilir. Bu sorunla başedebilmek için ridge regresyon BLUP, LASSO, elastic net, Bayesian yaklaşımları (BayesA, BayesB, BayesCπ, BayesDπ), destek vektör (support vector) regresyonu başta olmak üzere çok sayıda istatistiksel yaklaşım önerilmiştir. Bu yaklaşımlar hepsi (p>>n) sorunu ideal yaklaşımlardır. Ancak bu yaklaşımlar sözkonusu fenopit ile genomik seti arasında doğrusal bir ilşki olduğunu, başka bir ifade ile fenotipin SNP belirteşlerinin doğrusal bir fonksiyonu olduğu varsayılmaktadır. Bu yaklaşımlar fenotip ile genomik seti arasındaki doğrusal olmayan ilişkiyi yakalayamamaktadır. Doğrusal ilişki ile birlikte interaksiyon, epistatis gibi doğrusal olmayan ilişkilernin de modele dahil edilmesi kompleks fenotipik özellikler için ayrı bir önem taşıyabilir. GS amaçlı yaklaşımlarda bu sorun ile başedebilmek için ilk olarak 2011 yılında Okut ve arkadaşları tarafından yapay sınır ağları kullanılması önerilmiştir. Okut ve arkadaşları farelerde yoğun moleküler bilgi kullanılarak vücut kitle indeksi (BMI) için GS amaçlı çok katmalı regularize edilmiş tam bağlantılı yapay sinir ağları mimarisini (MLANN) önermişlerdir. Bu çalışmadan sonra derin öğrenme öğrenim algoritması kullanan daha kompleks yaklaşımlar GS amaçlı kullaılmaya başlanmıştır. Çok miktarda değişik derin öğrenme algoritmaları bulunmakta ve GS uygulamaları için her birinkendine özgü avantajlar sunmaktadır. Bu çalışmada, tam bağlantılı derin öğrenme yapay sinir ağları (DL-MLANN), evrişimli sinir ağları (CNN), tekrarlayan sinir ağları (RNN) ve uzun- kısa-süreli bellek (LSTM) yapay sinir ağları olmak üzere dört farklı derin öğrenme algoritmasinin tanımı yapılmıştır.

Introduction

Genomic selection (GS) or genomic prediction (GP) is a technique for predicting complex traits from observed genomic information such as SNPs, or whole genome sequence (Pérez-Enciso and Zingaretti, 2019). The fundamental concept in GS is to use functional genomic variation in DNA (genome-wide variation in“SNP markers”) together with phenotypes from an observed population to make prediction for the phenotypic responses of an untested (unobserved) population (Montesinos-López, 2021). The first time GS was described by Meuwissen et al. (2001) in dairy cattle to help breeders to identify genetically superior animals. The aim was to capture specific genes across the whole genome that are associated with favorite traits such as milk production, fertility, etc.

The rapid-growing commonly using the genomic selection since the last decade can be attributed to the decreasing in genotyping expenses, producing large number of DNA polymorphism (Wang et al., 2020). This, on the other hand, has created the problem a most common problem “big p, small n, aka p>>n” when making prediction in complex phenotypes using DNA polymorphisms (SNP markers). To address this problem, Meuwissen et al. (2001) introduced three statistical approaches; i) ridge regression BLUP (rrBLUP) which uniformly apply shrinkage on the SNP (marker) effects around zero, ii) BayesA which is a mixed model that assumes each marker is a random effect, and iii) BayesB which a general extension of

BayesA by setting a large proportion of SNP markers to be related with null effects. Both BayesA and BayesB differentially apply shrink on the marker effects but BayesB also execute the variable selection on the set of SNP markers. After Meuwissen et al. (2001) presented their study, many alternative approaches have been suggested to be useful for GS. Some of them are: Naïve and Scaled Elastic Net, Least Absolute Angle and Selection Operator (LASSO), Support Vector Regression (SVR) with a linear kernel, Bayesian LASSO (BL), BayesCπ, and BayesDπ (Azodi et al., 2019). Although these methods reveal good productive ability well when handling high dimensional data structure (i.e. $p>>n$). These methods all are based on a linear approximation from genetic polymorphism to phenotypes, and therefore fail to capture non-linearity of markers effects such as gene-gene interactions, epistasis and dominance effects, which are probable play important role to explain some variability of complex traits (Monir and Zhu, 2018; Azodi et al., 2019). To deal with p>>n problem and linearity, many non-linear methods have been applied to GS of complex traits (Figure 1). No single method performs best in all cases when efforts comparing the predictive ability of linear and non-linear approaches for the GS of complex traits. The predictive ability of both non-linear as well as linear algorithms depending on many issues including the number of individuals in the training data set, the number of marker and type of marker, the contribution of genetic to trait (heritability), effective population size, the genetic architecture and the quantity of associated loci (Azodi et al., 2019).

After the improvements in computational resources, the use of ANN algorithms in the field of GS started to gain momentum. As a member of ML methods, ANN is a computational approach that mimics human brain the way works the nerve cells perform. Artificial neural networks (ANNs) use different learning algorithms that can adaptively and independently adjust - or learn, in a sense - as they receive new input. This property of ANN makes them an attractive and efficient instrument to perform the non-linear statistical approach for complex traits (Anonymous, 2021). The first application of ANNs for GS was introduced by Okut et al. (2011) using fully connected ANNs to predict body mass index (BMI) in mice (i.e. an ANN with one input, one hidden and one output layer (aka, three layers) that are feed forward and interconnected. The first layer of proposed ANN by Okut et al. (2021) consists of input SNPs. The inner layer is hidden and is formed by neurons which adjustable and adaptively change the information received from the previous layer a series of transformations which in turn will pass the output of hidden layer as input to the last layer (output layer). Since then, more complex ANN architectures have been suggested for GS of the complex traits. In terms of connecting neurons into a network, there are three common type of architectures in conventional and DL; Feed-forward, convolutional and recurrent neural network. The term ‘‘deep” in DL denotes to the number of layers through which the data is passed. Conventional neural networks only comprise a couple (usually one to three) hidden layers, whereas DL networks can have as many as a couple hundred layers. Each hidden layer in DL networks consists multiple neurons with proposed architectures such as recurrent neural networks (RNN), long-short term memory (LSTM), convolutional neural network (CNN) and feed forward multilayer perceptron (MLP) has the potential for effectively using in GS (Alkhudaydi et al., 2019; Sandhu et al., 2021).

The input layer of any DL architecture for GS includes DNA (SNP) material, whereas the output layer consists of outcome variable of complex traits, with different number of

hidden layers. Unlike conventional neural networks, the algorithms used by DL considers many inner layers (hidden layers) between input and output during the training of network which feed the input (i.e. SNPs information) into a more abstract illustration at each stacked layer. Use of more than two or more hidden layers in DL can disclose non-linear associations between input data and response variables and can execute extremely complex functions (Maldonado et al., 2020). Here, in this paper, four different algorithms of DL for GS have been reviewed to provide potential application them for GS. The review will be started to introduce the DL Multilayer Layer Feed Forward Artificial Neural Network (DL-MLANN). Then Convolutional Neural Network (CNN) will be discussed. After then the Recurrent Neural Network (RNN) and Long-Short Term Memory (LSTM) neural networks will be discussed, respectively.

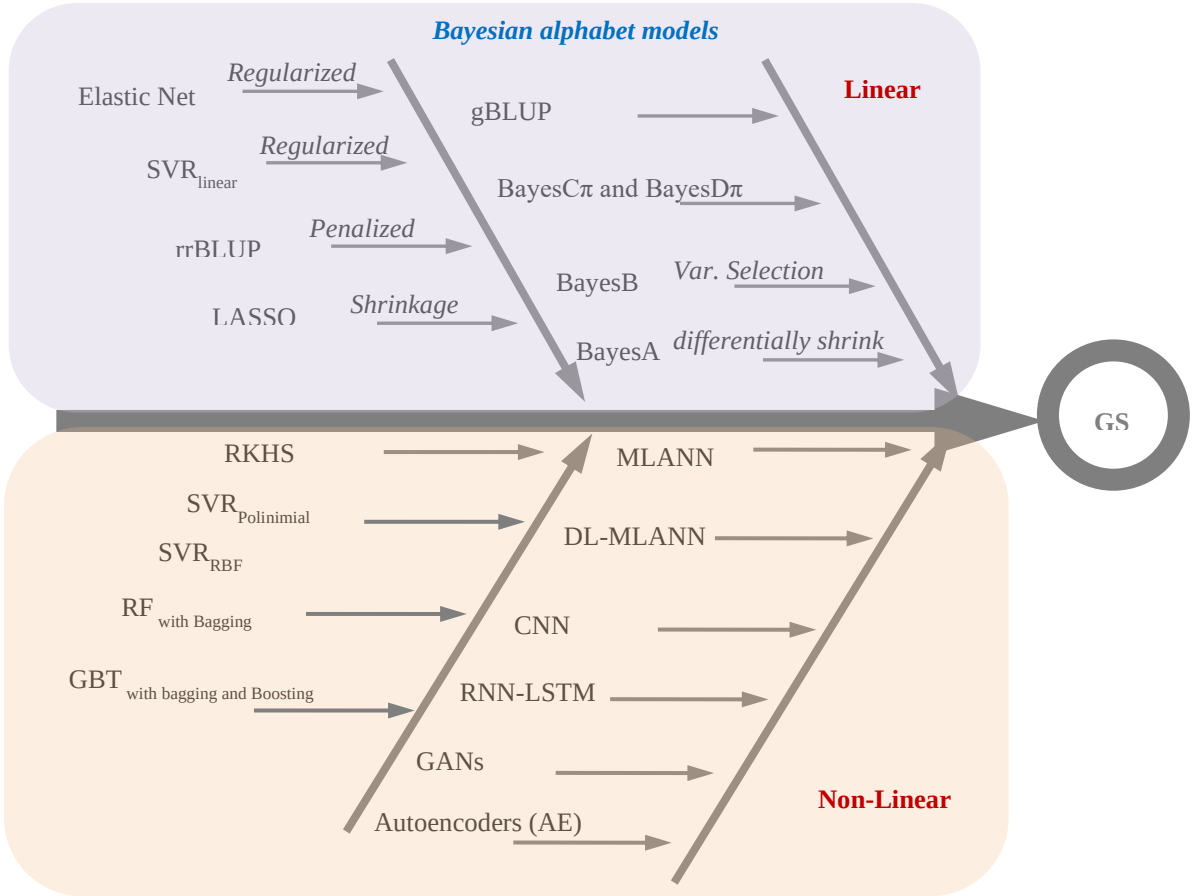

Figure 1. Some linear, non-linear and artificial neural networks (ANNs) algorithms applied in GS. Ridge regression best linear unbiased predictor(rrBLUP); reproducing kernel Hilbert spaces regression (RKHS); Bayesian ridge regression (BRR); support vector regression (SVR); random forest (RF); gradient tree boosting (GBT); multilayer artificial neural network (MLANN); deep learning multilayer artificial neural network (DL-MLANN); recurrent neural network(RNN); long-short term memory (LSTM); convolutional neural network (CNN) and generative adversarial network (GAN).

Şekil 1. GS'de uygulanan bazı doğrusal, doğrusal olmayan ve yapay sinir ağları (YSA) algoritmaları. Ridge regresyonu en iyi doğrusal yansız tahmin edici (rrBLUP); yenilenmiş kernel Hilbert regresyonunun (RKHS); Bayesian ridge regresyonu (BRR); destek vektör

regresyonunu (SVR); rastgele orman (RF); gradyan ağaç boosting (GBT); çok katmanlı yapay sinir ağı (MLANN); derin öğrenme çok katmanlı yapay sinir ağı (DL-MLANN); tekrarlayan sinir ağı(RNN); uzun-kısa süreli bellek (LSTM); evrişimsel sinir ağı (CNN) ve üretken çekişmeli ağ (GAN).

Deep Learning Multilayer Feedforward Artificial Neural Networks (DL-MLANN)

The multilayer feedforward deep neural network is well-known and the most popular used ANN paradigm in many real-world practices. The DL-MLANN is fully (densely) connected and separated into layers. In the left-most layer, there are input (independent) variables. Then input layer is followed by two or more hidden layers, which each of hidden layer consists certain number of number of neurons S neurons), and there is a bias specific to each neuron. The number of neurons in each hidden layer *i* can be different or the same. These neurons are connected only to neurons of the next layer (layer i + 1), and all the connection edges have different weights (i.e., $W^{(1)}$, $W^{(2)}$,…., $W^{(6)}$). This implies that there are no interconnections between neurons within the same layer, and that there are also no connections that convey data back from a higher layer to a lower layer connection (Figure. 2). Figure 2 shows a Deep-Multi-Layer Perceptron (DL-MLANN) diagram with multiple hidden layers (here, five hidden layers) and an assembly of SNPs as input and demonstrates a basic "neuron" with n inputs. One neuron is the output of applying a nonlinear activation function of (X_i, W_i, and biases *b*. Each neuron in the hidden layer performs a weighted summation of the inputs prior to activation which is then passed to a nonlinear activation function. The DL-MLANN is the most straightforward network to train. Like many nonparametric and parametric approaches such as smoothing splines (general additive model, thin-plate regression) and certain kernel regressions, DL-MLANN can also introduce overfitting (in essence with highly dimensional data structure, such as microarray data, genome wide association, GWAS, etc.) and consequential predictions can be outside the range of the training data set. The regularization approach (aka shrinkage) in MLANN allows bias of parameter approximates towards what are considered to be probable. In applications, the most common techniques of regularization in ANN are the early stopping methods and the Bayesian regularization (BR) (Okut, 2016).

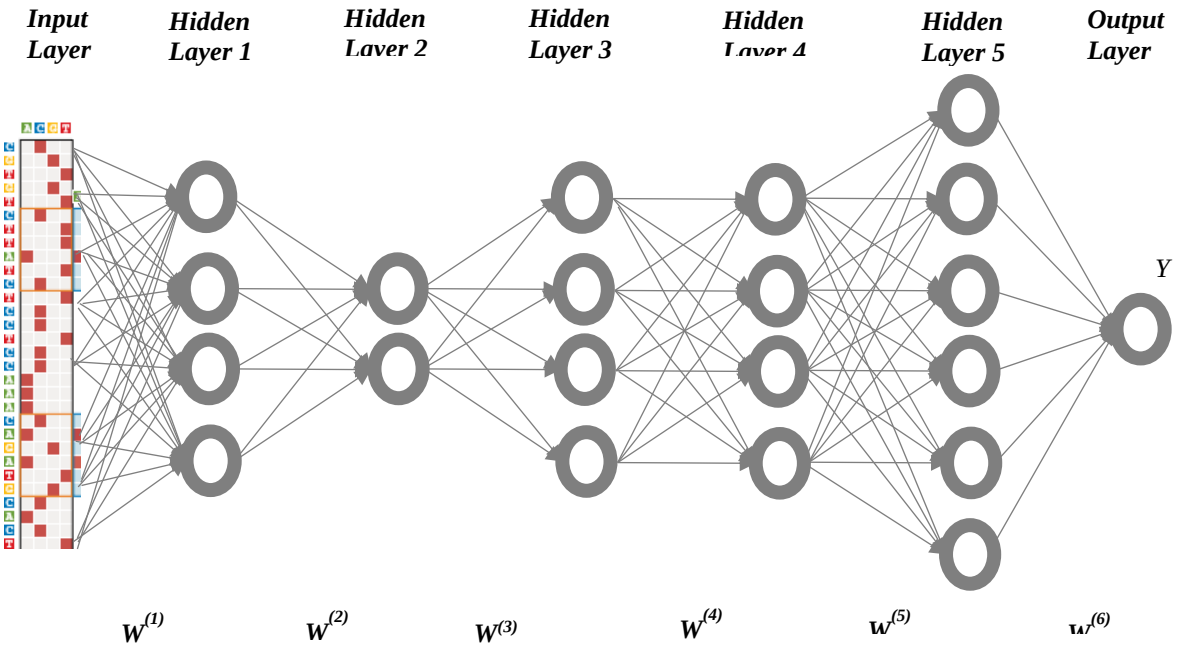

Figure 2. A 6-layer feedforward deep-multi-layer perceptron (DL-MLANN) neural network consist one input layer (most left), five hidden layers between input (most left with SNPs) and output layer (most right). The first, second, third, fourth and fifth hidden layer has three, two, four, four and six neurons, respectively. The output layer that corresponds to the phenotypes that needed to be predicted from the network architecture.

Şekil 2. A 6-katmanlı ileri beslemeli derin çok katmanlı algılayıcı (DL-MLANN) bir giriş katmanından oluşan sinir ağı (en solda), girişler arasında beş hidden katmanı (en solda SNP'ler) ve çıkış katmanı (en sağda). Birinci, ikinci, üçüncü, dördüncü ve beşinci gizli katman sırasıyla üç, iki, dört, dört ve altı nörona sahiptir. Ağ mimarisinden tahmin edilmesi gereken fenotiplere karşılık gelen çıktı katmanı.

Convolutional Neural Network

In mathematics, a convolution is expressed as fundamental transform between two functions, where one of the functions has to be a kernel. Instead of using a full matrix multiplication, CNN uses convolution in the hidden layers. CNN mainly are based on the feed forward MLANN architecture. Layers in a CNN is a assemble of connected and adaptively adjustable neurons which able to transmit a signal from a neuron to another. The underlaying idea of this deep learning neural network model is to immensely compute and assemble feature maps inferring non-linear relationships between the input (SNP markers) signals and the targeted phenotype or output (Koumakis, 2020). CNN is one of the most popular for feature extraction among others, for feature selection, dimensional reduction and for classification of image portrays such as DNA motif. Full explanation for CNN is presented in Figure 3.

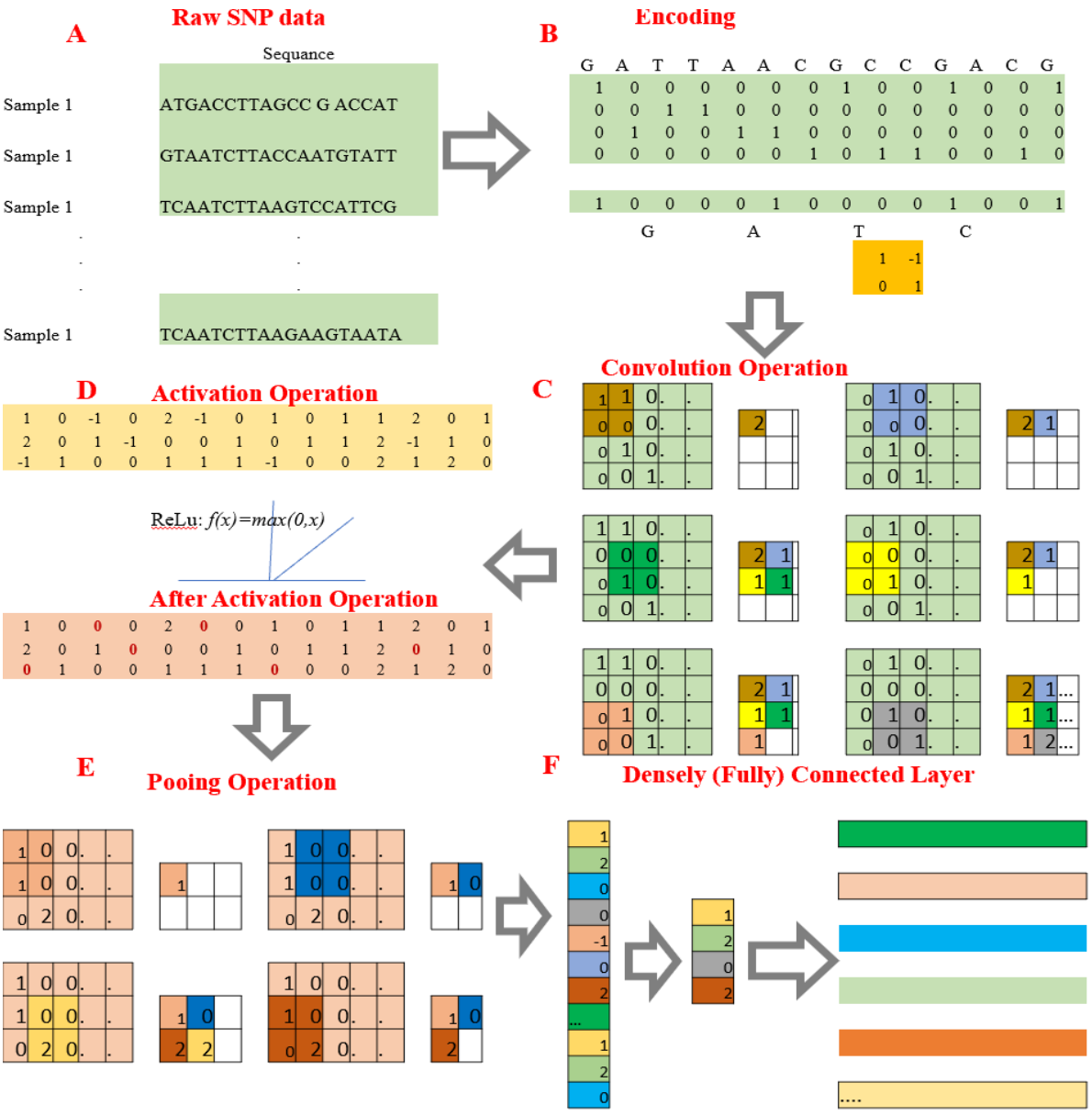

Figure 3. Steps of CNN for DNA motif in genomics. CNNs are collected of multiple layers of artificial neurons.

Şekil 3. Genomikte DNA motifi için CNN'nin adımları. Çoklu yapay nöron katmanlarından elde edilen toplam CNN'ler

In Figure 3, **A**) raw data after encoding of SNPs and then feeding not network and input matrix, **B**) filters are randomly initialized to each training DNA motifs at different resolutions, and the output of each convolved image is passed as the input to the following layer, **C**) encoded SNPs are convoluted on the source of initialized filters. Each filter is then multiplied by the corresponding input data through the sliding window, and sum is calculated and stored. Different stride is used for filter. Stride is number of pixels or the distance that the filter transfers over the SNP (input) matrix. Whereas stride values of two or greater is not very common, larger stride produces a smaller output. The filters in CNN are adaptively tunable parameters, commonly called network weight or network parameters, which are adapted in

the training progression. The sharing filters is one of the fundamental concepts of CNN to decrease the number of connections between each layer and thus reduce the risk of overfitting, **D**) the result from convolution layer is approximated (mapped) nonlinear technique using activation function in CNN so-called Rectified linear unit (ReLU). The ReLU nonlinear function allows for faster and more operative training by mapping negative values to zero and retaining positively resulted values, **E**) based on the feature map attained by convolution process, the pooling operation is conducted to further filter the feature map. Pooling simplifies the output by accomplishment nonlinear down sampling, reducing the effective number of parameters that the network requires to learn adaptively. The average pooling and max pooling are the two main pooling operations in CNN. The pooling operation in key step to decrease the curse dimensionality of the data information, the effective number of parameters needed to be estimate and the likelihood of overfitting. Multiple layers contained of convolution and pooling operations are stacked with each layer representing the data in slightly more abstract form than the previous layer. After many (about 20 or more) convolutional and pooling layers, the matrix created by pooling is flattened into vector and is fed it into a fully connected layer like a neural network. This last stage of CNN is a fully connected layer and is added as the output layer. In CNN architectures, fully connected layers can vary from one to multiple. Fully connected layers of CNN receive an input vector containing the flattened pixels of the DNA motif, which have been filtered, corrected and reduced by convolution and pooling layers. The softmax function is applied at the end CNN to the outputs of the fully (densely) connected layers. The steps from B to F are repeated all the input samples until the error stops declining. The error (distance) calculated between predicted and observed. To reduce the prediction error, well-known back propagation algorithm is conducted to compute the error gradient of all network weights. In particular, stochastic gradient descent (SGD) is frequently applied to update all filters to obtain a possible smallest the output error (Figure and explanation of figure partially adopted from Liu et al. (2020).

Deep Learning Recurrent Neural Network (RNN)

The feedforward MLANN can be extended for and effective and scalable neural network paradigm for several types of learning circumstances associated with serial (sequential) data. for use in sequential data or time sequences data. RNNs address time sequences data by leading multiple time steps that unfolds the network, adds new layers, and regauge the prediction error. These all processes end up in a very deep network. First, the connections between neurons in the hidden layer(s) form a directed or undirected graph along a time-based sequence allowing information to persevere. Through such a mechanism the recurrent neural networks memory is generated. An RNN architecture receives information from multiple preceding layers of the network (hidden state). Acquired information from the networks incorporate loops into the hidden layer. Loops in the network let information to stream multi-directionally so that the hidden state notifies the past information held along a temporal sequence. Therefore, the network types such as RNNs have an infinite dynamic response to sequential data. Figure 4A) depicts the structure of hidden layer for a RNN and

reveals the nonlinear activation function of the previous layers and the current input (p). In RNN, the hyperbolic tangent function is applied to generate the hidden state. The model has memory since the bias term in RNN hidden state is based on the "past". Consequently, the outputs from the preceding step are introduced as input to the current step. Therefore, an RNN has multiple copies with internal state of the same network, each passing a message to a successor (Figure 4B). Thus, the output value of the last time point is conveyed back to the neural network, so that the parameter estimation (weight calculation) of each time point is linked to the content of the previous time point.

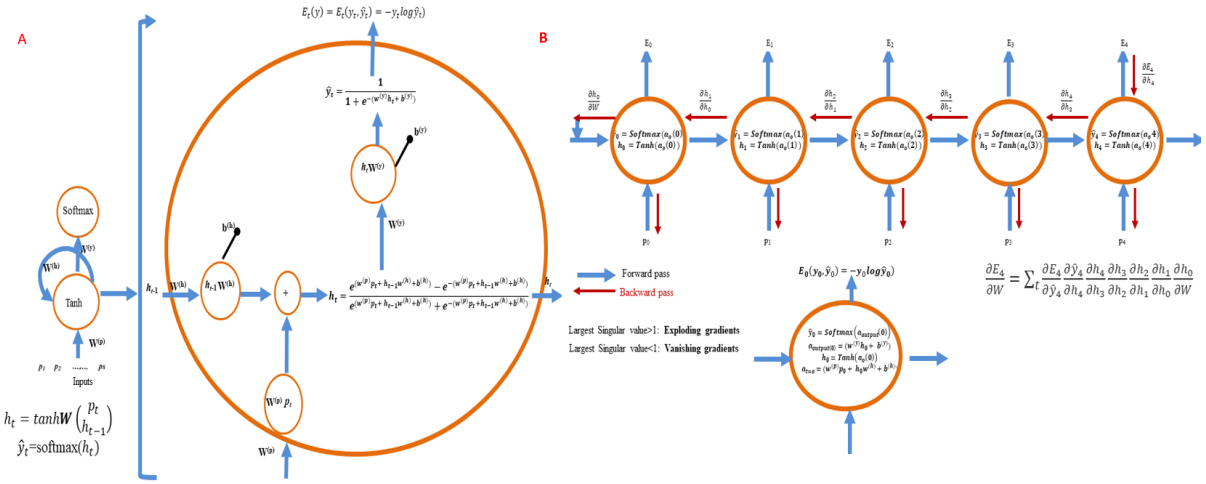

Figure 4. A typical RNN that has a hyperbolic tangent activation function
Şekil 4. Hiperbolik tanjant aktivasyon fonksiyonuna sahip tipik bir RNN

A typical RNN that has a hyperbolic tangent activation function $h_t = \left(\frac{e^{(x)}-e^{-(x)}}{e^{(x)}+e^{-(x)}}\right)$ to generate the internal hidden state (memory). Because of the internal hidden state in RNNs have a "memory" that information has been calculated so far is captured. The information in hidden state transmitted further to a second activation function $\hat{y} = \left(\frac{1}{1+e^{-(x)}}\right)$ to generate the predicted (output) values. In RNNs, the network weight (W) calculation of each time step of the network model is associated to the content of the earlier time point. We can process a sequence of vectors of inputs (p) by applying a recurrence formula at every time step (A). An unfurled RNN with a hidden state transmits relevant information from one input item in the series to others (B) and able to use the networks internal state to implement variable length order of inputs. The blue and red arrows in the figure are indicating the forward and the backward pass of the network, respectively. With backward pass, we sum up the contributions of each time step to the gradient. In other words, as W is used in every step up to the output, we need to backpropagate gradients from t = 4 through the network all the way to t = 0 (Adopted from Okut (2021)).

As in feedforward MLANNs, RNNs have two steps. These are a forward stage and a backward stage. Each works collectively throughout the training of the network. However, assemblies and calculation patterns differ. The forward stage typically has five steps: i) Summation step which combines two different source of information before nonlinear

activation function will be take place, ii) Applying hyperbolic tangent activation function to the summed of the two parameterized vectors ($W^{(p)}p_t + h_{t-1}W^{(h)} + b^{(h)}$) to force the output values between -1 and 1, iii) The network input to the output unit at time *t* with element-wise multiplication of output weights and with updated (current) hidden state ($h_t W^{(y)}$) iv). The calculation of the output of the network at time *t* and then applying of the activation function and v) Calculation of the error (loss function), $E_t(y_t, \hat{y}_t) = -y_t log\hat{y}_t)$ at each time step to start the "*backwarpass"*. After the forward stage (pass) of RNN, the calculated error (aka cost function) at each time step is injected backwards into the network to update the network weights at each iteration. The idea of RNN unfolding in Figure 4B takes place the bigger part in the way RNNs are implemented for the backward pass. Like conventional backpropagation in feed forward MLP, the backward pass consists of a repeated application of the chain rule. Hence, the backpropagation algorithm used for an RNN architecture to update the network parameters is called backpropagation through time (BPTT). In BPTT, the RNN network is unfolded in time to construct a feed forward MLP neural network. Then, the generalized delta rule is used to adaptively update the network parameters called weights $W^{(p)}$, $W^{(h)}$ and $W^{(y)}$ and biases $b^{(h)}$ and $b^{(y)}$ (Okut, 2021). Because the network parameters in a RNN are used in every step up to the output, we need to backpropagate gradients from last time step (*t*=*t*) through the network all the way to *t*=0. The Jacobians, $\left(\frac{\partial \boldsymbol{h}_j}{\partial h_{j-1}}\right)$, demonstrates the eigen decomposition given by $W^{(i)T} diag\left(f'(h_{j-1})\right)$, where the eigenvalues and eigenvectors are generated. Here the $W^{(i)T}$ is the transpose of the network parameters matrix (Okut, 2021). Subsequently, if the eigenvalue is greater or smaller than 1, the RNN suffers from vanishing or exploding gradient problems (see Figure 4).

Long Short-Term Memory (LSTM)

As expressed earlier, the output from RNNs is dependent on its aforementioned state or previous several (say N) time steps circumstances. If so, a typical RNN face difficulty in learning and sustaining long-range dependencies. Consider the unfolding RNN given in Figure 4B. Here, each time step of unfolded RNN requires a new copy of the network. With large RNNs, a couple thousands, even a couple millions of weights are needed to be updated using the chain rule ($\frac{\partial h_j}{\partial h_{j-1}}$). For example, as shown in Figure 4B, the derivative of four steps of RNN is $\frac{\partial h_4}{\partial h_3} = \frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial h_0}$. Imaging an unrolling the RNN a couple hundred times, in which every activation of the neurons within the network are replicated thousands of times. This signifying that, in particular for larger artificial neural networks millions of weights are needed. As Jacobian matrix will play a role to update the weights of network, the values for of the Jacobian matrix will vary between -1, 1 if tanh activation function is applied to the hidden neuron, $f(a_h(t)) = h_t = tanh\ (W^{(y)}h_t + b^{(y)})$. Then zero or around zero value will be expected for the derivatives of tanh (or sigmoid) activation function. Zero gradients drive other gradients in previous layers towards 0. Therefore, with small values in the Jacobian matrix and multiple matrix multiplications (t-j, in particular) the gradient values will be

shrunk exponentially fast, ultimately vanishing entirely after a couple time steps. As a result, the RNN ends up not learning long-range dependencies. As in RNNs, the vanishing gradients problem will be an important issue for the deep feedforward MLANN when multiple hidden layers (multiple neurons within each) are placed between input and output layers. The long short-term memory recurrent networks (LSTMs) are a special type of RNN that can deal with the vanishing gradient problem and can learn long-term dependencies. LSTM presents a memory unit and a gate mechanism for able to capture of the long dependencies in a sequence. The term "long short-term memory" in LSTM originates from the following insight. Simple RNN networks have long-term memory in the form of weights. The weights change adaptively throughout the training of the network, encoding general knowledge about the training data set. They also have short-term memory in the form of ephemeral activations, which flows from each node to successive nodes (Colah, 2021; Okut, 2021).

The network architecture for an LSTM block presented in Figure 5 reveals that the LSTM network extends RNN's memory and can selectively remember or forget information by structures called cell states and three gates. Therefore, in addition to a hidden state (memory) in RNN, an LSTM block typically has four additional layers. These are termed a cell state (C_t), an input gate (i_t), an output gate (O_t), and a forget gate (f_t). Each layer interrelates with another one in a very special way to generate information from the training data.

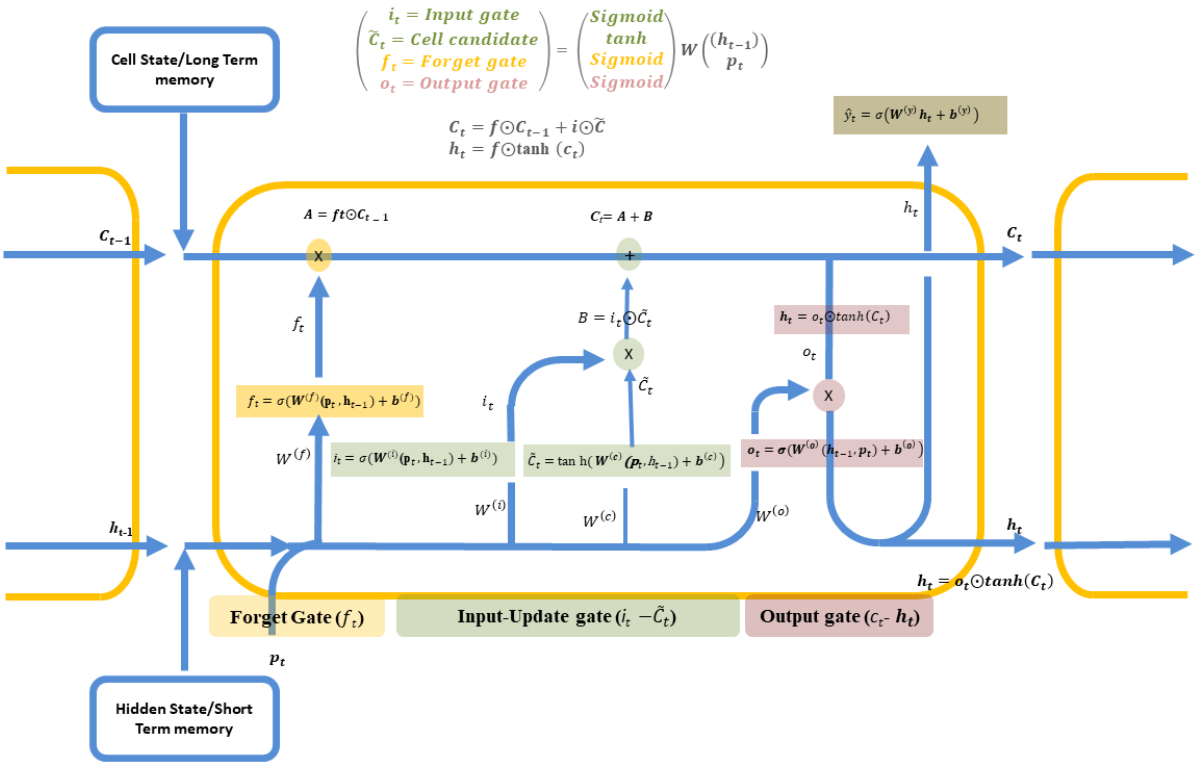

Figure 5. Illustration of LSTM block structure.
Şekil 5. LSTM blok yapısının gösterimi.

Illustration of LSTM block structure. Here ''⊙" symbolizes the element-wise multiplication. The C_{t-1}, C_t, h_t and h_t are previous cell state, current cell state, current hidden state and previous hidden state, respectively. The f_t; i_t; o_t are the values from forget, input and

output gates, respectively. The $\tilde{C}_t$is the candidate value for the cell state, $W^{(f)}$, $W^{(i)}$, $W^{(c)}$, $W^{(o)}$ are weight matrices consist of forget gate, input gate, cell state and output gate weights, and $b^{(f)}$, $b^{(i)}$, $b^{(c)}$, and $b^{(o)}$ are bias vectors associated with them.

The cell state is the crucial to LSTMs and characterizes the memory of LSTM networks. The process for the cell state is very much like to a conveyor belt or production chain. The information about the parameters runs straight forward the entire chain, with only certain linear interactions, such as multiplication and addition. The state of information depends on these interactions. If there are no interactions, the information will run along without changes. The *Forget Gate* ($\boldsymbol{f_t}$) in LSTM decides which information that should be thrown away or kept from the cell state. A sigmoid activation function takes place in forget this activation function, by nature, outputs values between 0 and 1 coming from the weighted input (W_f p_t), previous hidden state (h_{t-1}), and a bias (b_f). The equation of forget gates in Figure 5 can be rewritten as $f_t = \sigma(W^{(f)}(p_t, h_{t-1}) + b^{(f)}) = \frac{1}{1+e^{-(W^{(f)}.(p_t, h_{t-1})+b^{(f)})}}$. Here, σ indicates the sigmoid activation function, $^{(f)}$ and $\boldsymbol{b}^{(f)}$ are the weight matrix and bias vector, which will be learned from the input training data.

The *Input Gate* ($\boldsymbol{i_t}$) in LSTM networks controls what new information will be added to the cell state from the current input. This gate also undertakes the role to keep the memory contents from perturbation by irrelevant input. The input-update gate decides what new information should be kept in the cell state, which has two parts: a sigmoid layer and a hyperbolic tangent layer. The sigmoid layer is called the "input gate layer" because it decides which values should be updated. The tanh layer is a vector of new candidate values $\tilde{C}_t$ that could be added to the cell state. The input state and cell candidate are combined to create and update the cell state. The linear combination of the input gate and forget gate are used for updating the previous cell state (C_{t-1}) into current cell state (Ct). Here the input gate (i_t) governs how much new data should be taken into account via the candidate ($\tilde{C}_t$), while the forget gate (f_t) reports how much of the old memory cell content (C_{t-1}) should be retained ($C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$).

The *Output Gate* (o_t) regulates the type information to be revealed from the updated cell state ($\boldsymbol{C}_t$) to the output in a single time step. That is the output gate controls what the value of the next hidden state should be in each time step. As shown in Figure 5, the hidden state contains information on previous inputs. The calculated value of the hidden state for the given time step is conducted for the prediction ($\hat{y}_t = \text{softmax}(.)$). Here, softmax is a nonlinear activation function (sigmoid or hyperbolic tangent). The final outcomes of the output gate is an adaptively updated of the hidden state, and this is used for the prediction at time step *t*. Therefore, the output gate does the assessment regarding what portions of the cell state (C_t) is offered in the hidden state (h_t). The new cell and new hidden states are then transmitted to the next time step. The information up to her summarized the forward pass of LSTM. The forward pass can be summarized into 7 step and given below.

1. *Forget gate*: Controls what information to throw away and decides how much from the part should be remember. $f_t = \sigma(W^{(f)}(\mathrm{p}_t, \mathrm{h}_{t-1}) + b^{(f)})$

2. *Input-Update Gate:* Controls information to add cell state from current input and decides how much should be added to the cell state $\iota_t = \sigma(W^{(i)}(\mathrm{p}_t, \mathrm{h}_{t-1}) + b^{(i)})$,$\tilde{C}_t = \tanh(W^{(c)}(\mathrm{p}_t, \mathrm{h}_{t-1}) + b^{(c)})$
3. *Output gate:* Determines the part of the current cell state makes it *to the output* $o_t = \sigma(W^{(o)}(h_{t-1}, p_t) + b^{(o)})$.
4. *Current cell state:* $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$
5. *Current hidden state:* $h_t = o_t \odot tanh(C_t) \Rightarrow h_t = LSTM((p_t, h_{t-1})$
6. *LSTM block prediction:* $\hat{y}_t = \sigma\left(W^{(y)} h_t + b^{(y)}\right)$
7. *Calculate the LSTM block error for the time step:* $E_t(y_t, \hat{y}_t) = -y_t log \hat{y}_t)$

The forward pass of RNN and LSTM is illustrated in Figure 6. As in the RNN networks, an LSTM network generates an output ($\hat{y}_t$) at each time step that is used to train the network via gradient descent. During the backward pass, the network parameters are updated at each epoch (iteration). As in RNN, the total error is calculated by the summation of error from all time steps. Details for the backward pass of RNN and LSTM can be found in Okut (2021).

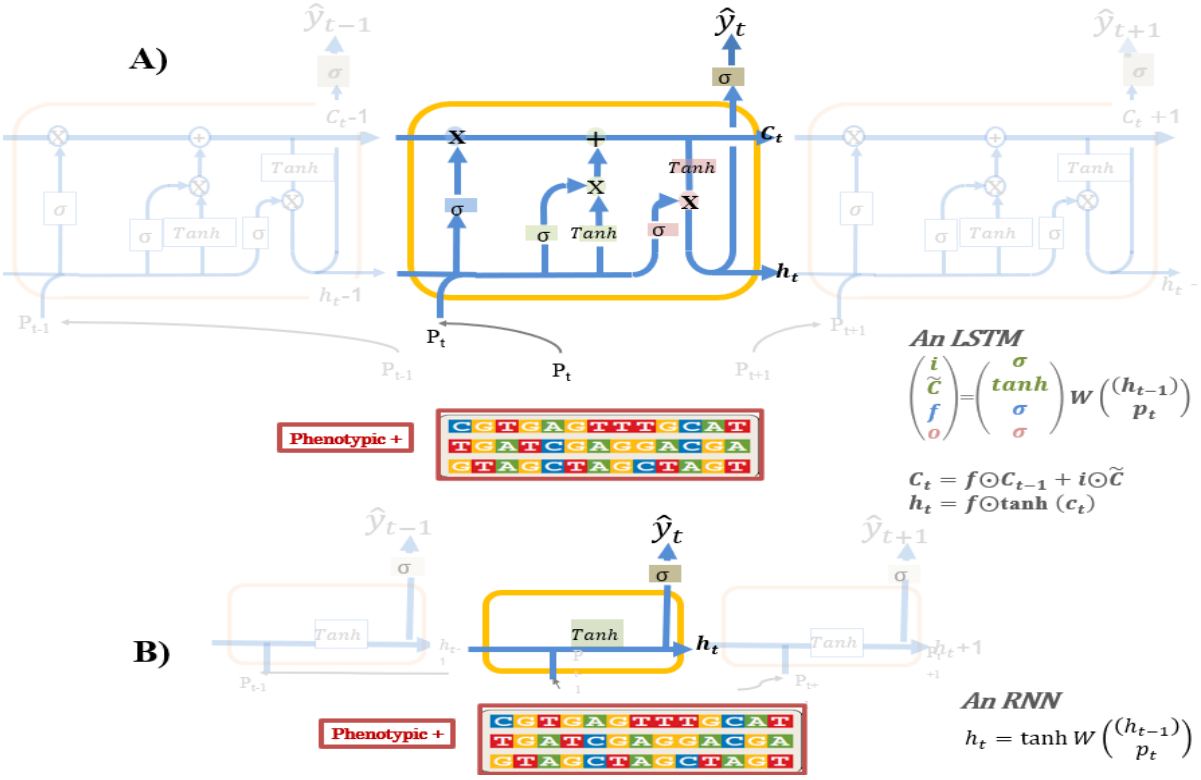

Figure 6. An LSTM unit from 3-time steps with input data (Phenotypic and SNPs).
Şekil 6. Veri girişli 3 zaman aşamalı bir LSTM birimi (Fenotipik ve SNP'ler)

A) LSTM network takes inputs from to the current time step to update the hidden state and ($LSTM((p_t, h_{t-1}))$ with relevant information. The "X" in the circles denote point-wise operators, σ and tanh are sigmoid $\left(\frac{1}{1+e^{-(x)}}\right)$ (generates a value between 0 and 1) and

hyperbolic tangent, $\left(\frac{e^{(x)}-e^{-(x)}}{e^{(x)}+e^{-(x)}}\right)$ (generates a value between -1 and 1), respectively. B) An RNN network with 3- time steps. It has only a tangent $\left(\frac{e^{(x)}-e^{-(x)}}{e^{(x)}+e^{-(x)}}\right)$

Conclusion

Deep learning algorithms can perform complex processes efficiently, while ML algorithms cannot. A deep learning neural network contains a wide variety of algorithms that depend on numerous hyperparameters. Despite potential advantages of deep learning networks, they introduce some difficulties in its implementation. One of the main difficulties is that the deep learning algorithms is predisposed to to overfitting. Moreover, deep learning algorithms need very large datasets for the training part. Therefore, these may not be available in all genomic prediction settings. Another limitation of the deep learning algorithms is the interpretability of their results. CNNs in particular, appear as the most promising predictive tool for genomic selection. This could be due in part to the fact that convolutional filters may capture some functional sequence motifs.

References

Alkhudaydi, T., Reynolds, D., Zhou, J., Iglesia, B., and Griffiths, S., 2019. An exploration of deep-learning based phenotypic analysis to detect spike regions in field conditions for UK bread wheat. Plant Phenom.7368761. DOI: 10.34133/2019/7368761.

Azodi, BC., McCarren, A., Roantree, M., de los Campos, G. and Shiu, SH., 2019. Benchmarking Parametric and Machine Learning Models for Genomic Prediction of Complex Traits. G3-Genes, PMID: 31533955, PMCID: PMC6829122, DOI: 10.1534/g3.119.400498.

Colah, C. Understating LSTM Network 2021. https://colah.github.io/posts/2015-08-Understanding-LSTMs/ .

De los Campos G, Gianola D, Rosa GJM, Weigel KA, Crossa J., 2010 Semi-parametri genomic-enabled Prediction of genetic values using reproducing kernel Hilbert spaces methods. Genet Res. 92(4):295–308. Available from: http://dx.doi.org/10.1017/S0016672310000285.

Koumakis, L., 2020. Deep learning models in genomics; are we there yet? Computational and Structural Biotechnology Journal 18, 1466–1473. https://doi.org/10.1016/j.csbj.2020.06.017.

Liu, J., Li, J., Wang, H., and Yan, J. 2020. Application of deep learning in genomics. Sci China Life Sci 63, 1860–1878. https://doi.org/10.1007/s11427-020-1804-5.

Lipton, C. Z., Berkowitz, J. and Elkan, C. A Critical 2021. Review of Recurrent Neural Networks for Sequence Learning. arXiv:1506.00019v4**.**

Maldonado C, Mora-Poblete F, Contreras-Soto RI, Ahmar S, Chen J-T, do Amaral Júnior AT and Scapim CA., 2020. Genome-Wide Prediction of Complex Traits in Two Outcrossing Plant Species Through Deep Learning and Bayesian Regularized Neural Network. Front. Plant Sci. 11:593897. DOI: 10.3389/fpls.2020.593897.

Monir MM, Zhu J., 2018. Dominance and Epistasis Interactions Revealed as Important Variants for Leaf Traits of Maize NAM Population. Front Plant Sci. 18;9:627. Available from: http://dx.doi.org/10.3389/fpls.2018.00627.

Okut H, Gianola D, Rosa GJM, Weigel KA., 2011. Prediction of body mass index in mice using dense molecular markers and a regularized neural network. Genet Res. 93(3):189–201. Available from: http://dx.doi.org/10.1017/S0016672310000662

Okut H., 2016. Artificial Neural Networks Model and Application. Joao Juis G. Rosa (Eds), Bayesian Regularized Neural Networks for Small n Big p Data (pp 27-48). London, UK. IntechOpen.

Okut H., 2021. Deep Learning and Application, Pier Luigi Mazzeo and Paolo Spagnolo, (Eds), Deep Learning for Subtyping and Prediction of Diseases: Long-Short Term Memory (pp 27-48). London, UK. IntechOpen. DOI: 10.5772/intechopen.96180.

Sandhu KS, Lozada DN, Zhang Z, Pumphrey MO and Carter AH., 2021. Deep Learning for Predicting Complex Traits in Spring Wheat Breeding Program. Front. Plant Sci. 11:613325. DOI: 10.3389/fpls.2020.613325.

Wang, H., Cimen, E., Singh, N., and Buckler, E., 2020. Deep learning for plant genomics and crop improvement. Curr. Opin. Plant Biol. 54, 34–41. DOI: 10.1016/j.pbi.2019.12.010.