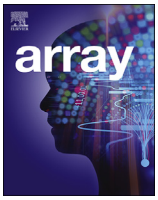
Array 11 (2021) 100068

Contents lists available at ScienceDirect

Array

journal homepage: www.elsevier.com/journals/array/2590-0056/open-access-journal



Deep learning regularization techniques to genomics data

Harouna Soumare[a,b,*], Alia Benkahla[b,1], Nabil Gmati[c,1]

[a] The Laboratory of Mathematical Modelling and Numeric in Engineering Sciences, National Engineering School of Tunis, University of Tunis El Manar, Rue Béchir Salem Belkhiria Campus Universitaire, B.P. 37, 1002, Tunis Belvédère, Tunisia
[b] Laboratory of BioInformatics, BioMathematics, and BioStatistics, Institut Pasteur de Tunis, 13 Place Pasteur, B.P. 74 1002, Tunis, Belvédère, Tunisia
[c] College of Sciences & Basic and Applied Scientific Research Center, Imam Abdulrahman Bin Faisal University, P.O. Box 1982, 31441, Dammam, Saudi Arabia

ARTICLE INFO

Keywords:
Deep learning
Overfitting
Regularization techniques
Dropout
Genomics

ABSTRACT

Deep Learning algorithms have achieved a great success in many domains where large scale datasets are used. However, training these algorithms on high dimensional data requires the adjustment of many parameters. Avoiding overfitting problem is difficult. Regularization techniques such as L^1 and L^2 are used to prevent the parameters of training model from being large. Another commonly used regularization method called Dropout randomly removes some hidden units during the training phase. In this work, we describe some architectures of Deep Learning algorithms, we explain optimization process for training them and attempt to establish a theoretical relationship between L^2-regularization and Dropout. We experimentally compare the effect of these techniques on the learning model using genomics datasets.

1. Introduction

In the last decade, Deep Learning (DL) algorithms have achieved tremendous success in many domains where large scale datasets are used such as Bioinformatics [2,13,50,62,88,94], Natural Language Processing [5,15,28,47,71], Computer Vision and Speech Recognition [1,4,29,34,37,56,65].

In this work, we review a class of DL algorithms called Feedforward Neural Network (FNN) [54,68,91], in which information moves in one direction, from input to output through sequential operations called "layers". These models are the generalization of logistic regression models, both (FNN and logistic regression) are widely used in Bioinformatics and Biomedical science to perform classification and diagnosis tasks [8,20,21,24,27,44,48,70,73]. In most cases, we look for a non linear mapping $y = f(\mathbf{x})$ between a variable y and a vector of variables $\mathbf{x}$. The form of f depends on the complexity of the studied problem.

Logistic regression defines a low complexity model using a simple non linear mapping from inputs to outputs. Whereas FNN defines a more complex mapping between inputs and their corresponding outputs, thus resulting models have high complexity and flexibility and better prediction capacity. However, increasing the complexity of predictive models increases also the risk of overfitting problem, which repercussion is that the training model fits well the training dataset but looses its prediction capacity on unseen datasets.

Preventing overfitting problem is one major challenge in training these algorithms. However, there are many techniques that deal with the problem of overfitting called "regularization techniques". The most used regularization techniques in Machine Learning (ML) community are L^1 and L^2 regularization's [53]. The idea is to prevent the weights of the model from being large by adding a supplementary term to the loss function. The effect of this penalization is to make it so the learning algorithm prefers to learn small weights. This method makes models less complex and avoid the risk of overfitting. Another commonly used regularization technique so-called "Dropout", developed by Hinton et al. [33] consists to randomly remove some neurons (in hidden layers) during the training phase. This forces the hidden units to extract useful information's from the input data and reduce co-adaptation between hidden units, thus making the model less sensitive to the specific weights of neurons. The Dropout technique allows to train an exponential number of (thinned) Networks in a reasonable time [33]. During the test phase, taking the mean prediction of the different (thinned) Networks is equivalent to test on a single Network with all the hidden neurons [6] (without dropping out any unit). To compensate the fact that the weights are learned under Dropout, the outcome weights of neurons of each

[*] Corresponding author. The Laboratory of Mathematical Modelling and Numeric in Engineering Sciences, National Engineering School of Tunis, University of Tunis El Manar, Rue Béchir Salem Belkhiria Campus Universitaire, B.P. 37, 1002, Tunis Belvédère, Tunisia.
E-mail addresses: soumare.harouna@enit.utm.tn (H. Soumare), Alia.Benkahla@pasteur.tn (A. Benkahla), nmgmati@iau.edu.sa (N. Gmati).

[1] These autors contributed equally, the order of their names is alphabetical.

https://doi.org/10.1016/j.array.2021.100068
Received 28 November 2020; Received in revised form 26 March 2021; Accepted 10 May 2021
Available online 24 May 2021

hidden layer are multiplied by the Dropout rate of that layer, which is a gain in terms of computation time. However, the quality of this approximation remains little known.

Many theoretical Dropout analyses have been explored [6,23,31,49,55,58,75,79,81]. Baldi et al. [6] showed how the technique acts as adaptative stochastic Gradient Descent. Wager et al. [79] analyzed Dropout as an adaptive regularizer for Generalized Linear Models (GLMs). Ma et al. [46] attempted to explicitly quantify the gap between Dropout's training and inference phases and showed that the gap can be used to regularize the standard Dropout training loss function.

This paper explains the mathematics behind training DL algorithms and attempts to further establish the theoretical relationship that exists between Dropout and other regularizations, mainly L^2 norm. We compare experimentally the effects of regularization techniques on training models using two different genomic classification datasets.

The human DNA is a long chain of 3 billion base pairs, the function of a large part of it, is unknown. Some fragments of DNA called genes code for proteins that play important roles in chemical processes essential to life. Some changes in the genes cause a dysfunction in the production of the corresponding proteins, which could cause genetic diseases. The most common genetic changes are called Single Nucleotide Polymorphisms (SNPs) and are caused by a change of a base pair by another one at a given position in the genome. It has been shown that some SNPs are involved in several human diseases and can be used to predict human reponse to certain drugs [27].

In our experiments, we started by using Logistic Regression on cancer datasets, obtained from the Expression Project for Oncology (EXPO) [60]. Then we trained FNN on one 1000 Genomes Project dataset for individual ancestry prediction according to their genetic profile [59]). All individuals are represented in both datasets by their SNPs profile [14].

This work is organized as follows:Section 2 describes the FNN architectures and the mathematics behind them; in Section 3 Gradient descent algorithm is presented; Section 4 describes the traditional regularization techniques and Dropout, Section 5 describes the materials and methods and in Section 6, we present the experimental results, where different regularization techniques are used.

2. Deep Learning: Feedforward Neural Network(FNN)

In this work, we discuss Feedforward Neural Network (FNN) [42,54,57,69,76,91] or Multi-Layer Perceptron (MLP). In such Networks, the information moves only from the input to the output (see Fig. 2), without any loop. This type of model is mostly used for supervised ML tasks such as regression or classification tasks, where the target function is known. The basic supervised learning algorithm is linear regression [12,51,82], in this task the algorithm learns to map an input data $\mathbf{x} \in \mathbb{R}^d$ to some real value y, by a linear transformation

$$\begin{aligned} f:\ & \mathbb{R}^d \to \mathbb{R} \\ & \mathbf{x} \to z = \mathbf{x}\cdot\mathbf{w} + b \end{aligned}.$$

Where $\mathbf{w}$ and b are respectively the weight vector and the bias term. The symbol "·" is the dot product between two vectors. Another simple supervised learning algorithm called logistic regression is used for the classification problems where the target function takes discrete values. Given an input data $\mathbf{x}$, the logistic regression [18,19,39,74,86] applies a non linear function to its corresponding linear regression output z, to produce classes membership probabilities. For example, in a binary classification task, given $\mathbf{x}$ and it corresponding class C_1, logistic regression algorithm outputs the conditional probability $P(C_1|\mathbf{x})$ of $\mathbf{x}$ given C_1. This probability is given by sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$. In the case where there are more than two classes, the conditional probability $P(C_i|\mathbf{x})$ is given by the softmax function $softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{n_c} e^{z_k}}$. Where $\mathbf{z} = (z_1\, z_2 \ldots z_{n_c})$ and $z_i = \mathbf{x}\cdot\mathbf{w}_i + b_i$. $\mathbf{w_i}$ and $\mathbf{b_i}$ are respectively the weight vector and bias term of the ith class C_i. n_c is the number of classes and the class probabilities sum up to 1, i.e. $\sum_{i=1}^{n_c} softmax(\mathbf{z})_i = 1$. Fig. 1 describes the simplest possible Neural Network (NN), which contains a single neuron corresponding exactly to an input-output mapping. A neuron with sigmoid output function is equivalent to logistic regression. FNN is a nonlinear function, which is also composed of several simpler functions(neurons, where the output of a neuron can be used as an input of another. Each of these functions provides a new representation of the input data. It is composed of an input layer, one or more hidden layer(s) and an output layer.

2.1. Supervised Neural Network

Let's consider an L hidden layer Feedforward Neural Network, in which n input training samples $X = \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ are labeled, i.e., given an input $\mathbf{x}_i$, the corresponding output by the model is known and denoted y_i or $\mathbf{y}(\mathbf{x}_i)$. Where $\mathbf{y}$ is a vector containing labels. A standard Neural Network can be described as follows:

$$a_j^{(l)} = \varphi\left(z_j^l\right), \tag{1}$$

$$z_j^l = \sum_i w_{ij}^l a_i^{(l-1)} + b_j^l = \mathbf{a}^{(l-1)}\cdot\mathbf{w}_j^l + b_j^l, \tag{2}$$

where z_j^l, b_j^l and a_j^l ($a_j^0 = x_j$, for a d-dimensional input $\mathbf{x} = (x_1\, x_2 \ldots x_d)^T$) are the jth hidden input, bias and activation function of the lth layer, respectively. w_{ij}^l is the weight connection from the ith unit of the $(l-1)$th layer to the jth unit of the lth layer. $\mathbf{w}_j^l$ and $\mathbf{a}^{(l-1)}$ are, respectively, the incoming weight vector to the jth neuron of layer l and the output vector of *(l-1)*th layer, φ is any activation function. FNNS can be seen as a generalization of simple regression models. In fact, keeping only the input layer with one linear output neuron in a Feedforward Network defines a linear regression, and with a sigmoid or softmax function at the output layer represents a logistic regression. Learning of a supervised Network [26,63,87,90] consists to find the parameters $\mathbf{w}_j$ and b_j so that output $\mathbf{a}^L$ from the model approximates the desired output vector $\mathbf{y}(\mathbf{x})$, for all training inputs $\mathbf{x}$. To achieve this goal, we define a mean squared error loss function

$$C = \frac{1}{n}\sum_{\mathbf{x}\in X} C_{\mathbf{x}}, \tag{3}$$

$C_{\mathbf{x}} = \frac{1}{2}\|\mathbf{y}(\mathbf{x}) - \mathbf{a}^L(\mathbf{x})\|_2^2 = \frac{1}{2}\sum_{k=1}^{n_c}(y_k - \mathbf{a}_k^L)^2$. Where y_k and $\mathbf{a}_k^L$ are kth output activation and desired output respectively, for a given input $\mathbf{x}$.

There is another choice of loss function known as crossentropy. To define this function, let's consider a binary classification problem with sigmoid output function $a^L(\mathbf{x}) = \sigma(z)$, for each training input sample $\mathbf{x}$, we have $P(y = 0|\mathbf{x}) = 1 - P(y = 1|\mathbf{x}) = 1 - a^L(\mathbf{x})$ and more generally

$$P(y = y_i|\mathbf{x_i}) = a^L(\mathbf{x_i})^{y_i}\left(1 - a^L(\mathbf{x_i})^{1-y_i}\right).$$

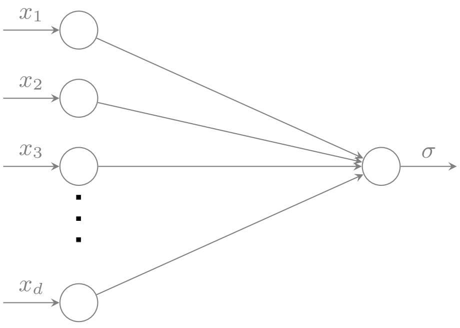


Fig. 1. Logistic regression "neuron".

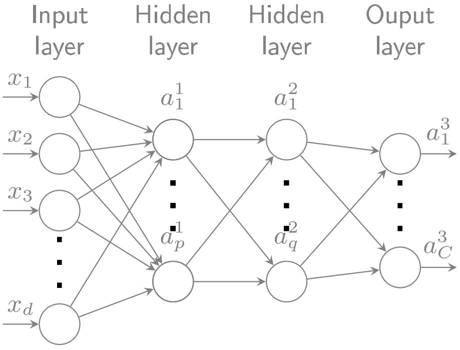


Fig. 2. Classification network.

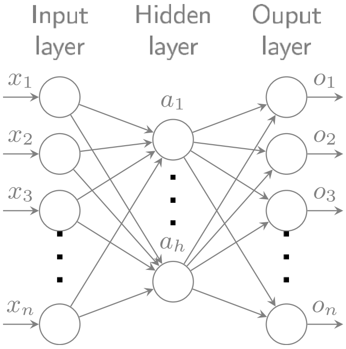


Fig. 3. Autencoder

Supposing that the couples $(\mathbf{x_i}, y_i)$, $i \in \{0, \dots, 1\}$ are independent, the likelihood function is given

$$P\left(y|X) = -\prod_{i=1}^{n} P(y=y_i|X\right) \tag{4}$$

$$= -\prod_{i=1}^{n} a^L(\mathbf{x_i})^{y_i}\left(1-a^L(\mathbf{x_i})^{1-y_i}\right). \tag{5}$$

Training the NN consists to maximize the likelihood function which is equivalent to minimize the crossentropy loss function defined by

$$C = -\frac{1}{n}\sum_{i=1}^{n} y_i \log a^L(\mathbf{x_i}) + (1-y_i)\log(1-a^L(\mathbf{x_i})). \tag{6}$$

Consider now, a multi-class classification problem, where the labels are mutually exclusive. In this case, (6) takes the form

$$C = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{n_c} y_k(\mathbf{x_i})\log a_k^L(\mathbf{x_i}). \tag{7}$$

$a_k^L(\mathbf{x_i})$ is the softmax function satisfying $0 \le a_k^L(\mathbf{x_i}) \le 1$ and $\sum_{k=1}^{n_c} a_k^L(\mathbf{x_i}) = 1$. In the rest of this work, C denotes the loss function defined by (3).

2.2. Unsupervised Neural Network(Autoencoder)

So far, we have described FNN in the supervised learning case. Here, we suppose that input samples $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ are unlabeled, where $\mathbf{x}_i \in \mathbb{R}^d$. Autoencoder is one the most used unsupervised learning algorithms [41,52,77,83,93]. An Autoencoder is a NN designed to learn an identity function in a way that the original input can be reconstruct from a compressed version. Such a network will allow the discovery of a more efficient and compressed representation of the input data. It consists of two parts, an encoder and a decoder. Encoder maps input samples to a hidden representation and decoder tries to reconstruct inputs from an encoder, so it contains at least one hidden layer. In an Autoencoder Network, the target $\mathbf{y}(\mathbf{x})$ of each input sample $\mathbf{x}$ is the input it self, i.e. $\mathbf{y}(\mathbf{x}) = \mathbf{x}$, $\forall \mathbf{x} \in \mathbb{R}^d$. At the end the output has the size of the input.

The main objective of an Autoencoder is to automatically capture the most relevant features from input data. It is also used as a nonlinear dimensionality reduction technique [32,66,80] to transform a high d dimensional data to a lower dimensional data. Mathematically it is defined by the following application:

$$\begin{aligned} o:\ & \mathbb{R}^d \rightarrow \mathbb{R}^d \\ & \mathbf{x}_i \rightarrow \varphi'_{W'^1} \circ \varphi_{W^1}(\mathbf{x}_i), \quad \forall \mathbf{x_i} \in \mathbb{R}^d \end{aligned}$$

Where φ_{W^1} and $\varphi'_{W'^1}$ are the encoding and decoding functions parametrized by $W^1 \in \mathbb{R}^{d\times h}$ and $W'^1 \in \mathbb{R}^{h\times d}$ respectively and defined as follow:

$$\begin{aligned} \varphi_{W^1}:\ & \mathbb{R}^d \rightarrow \mathbb{R}^h \\ & \mathbf{x}_i \rightarrow \mathbf{a}_h(\mathbf{x}_i) \end{aligned}, \quad \begin{aligned} \varphi'_{W'^1}:\ & \mathbb{R}^h \rightarrow \mathbb{R}^d \\ & \mathbf{a}_h(\mathbf{x}_i) \rightarrow o(\mathbf{x}_i) \end{aligned}.$$

Where $\mathbf{a}_h$ and o are, respectively, the hidden and output layers output vectors. The parameters (W^1, W'^1) are learned by minimizing the reconstruction error between the input and the output of Network

$$L = \frac{1}{2n}\sum_{i=1}^{n} \|\mathbf{x}_i - \varphi'_{W'^1} \circ \varphi_{W^1}(\mathbf{x}_i)\|_2^2.$$

After Autoencoder training, the decoding layers are removed and the encoding layers are retained and the learned matrix W^1 is then used as parameters of the first layer(s) of the supervised Network (see Fig. 3). Alternatively, the couple (W^1, W'^1) can be learned jointly(see Fig. 4) with the classification Network [22,61,92]by minimizing C_T, the following loss function

$$C_T = C + \frac{\gamma}{2n}\sum_{i=1}^{n} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|_2^2.$$

Where $\hat{\mathbf{x}}_i = \varphi'_{W'^1} \circ \varphi_{W^1}(\mathbf{x}_i)$, $\hat{X}$ is a matrix whose rows are formed by $\hat{\mathbf{x}}_i$'s and γ is a tuning parameter.

3. Gradient descent

Once the loss function is defined, gradient descent strategy is typically used to minimize it. Gradient descent is a first-order optimization strategy for nonlinear minimization problems [17]. The loss function C is minimized iteratively by using Gradient descent method [3] given by

$$w_{ij}^l \rightarrow w_{ij}^l - \frac{\alpha}{n}\sum_{\mathbf{x}\in X}\frac{\partial C_\mathbf{x}}{\partial w_{ij}^l}. \tag{8}$$

Where α is the learning rate. For the sake of simplification, we assume that there are no bias terms b_j^l or simply consider it as an additional component of $\mathbf{w}_j^l$. At each iteration, we have to compute partial derivatives of $C_\mathbf{x}$ for each training input $\mathbf{x}$, and then average them to update weights w_{ij}^l. Unfortunately, this method can be very expensive and learning occurs slowly when the number of training inputs is large. This problem of learning slowness can be avoided by the Stochastic Gradient

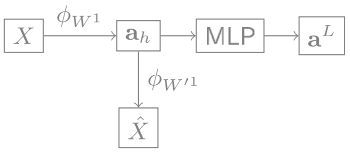


Fig. 4. Classification network & autoencoder.

Descent (SGD) method.

3.1. Stochastic gradient descent

The idea of stochastic gradient descent [9,10,40,89] is to estimate at each iteration partial derivatives for only a small randomly chosen sample $X_m = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x}_m\}$ called mini-batch and train with it.

$$w^l_{ij} \to w^l_{ij} - \frac{\alpha}{m}\sum_{\mathbf{x}\in X_m}\frac{\partial C_{\mathbf{x}}}{\partial w^l_{ij}}. \tag{9}$$

We then take another randomly chosen mini-batch and the weight parameters are updated on it, until the training inputs are exhausted, which is called an epoch of training. At this point, we start again with a new epoch. To compute the partial derivatives, $\frac{\partial C_{\mathbf{x}}}{\partial w^l_{ij}}$ at each layer, we apply the chain rule:

$$\frac{\partial C_{\mathbf{x}}}{\partial w^l_{ij}} = \delta^l_j(\mathbf{x})\frac{\partial z^l_j}{\partial w^l_{ij}} = \delta^l_j(\mathbf{x})a^{l-1}_i(\mathbf{x}).$$

Where $\delta^l_j = \frac{\partial C_{\mathbf{x}}}{\partial z^l_j}$ represent the error function of j neuron in the lth layer, for an input $\mathbf{x}$. For the sake of simplicity, we just write δ^l_j and a^{l-1}_i instead of $\delta^l_j(\mathbf{x})$ and $a^{l-1}_i(\mathbf{x})$. This expression tells us how a little change in the weighted input to the jth neuron in layer l changes the overall behavior of the loss function. The backpropagation algorithm is used to compute δ^l_j for each layer.

3.2. Backpropagation

Backpropagation [30,84,85] is a widely used algorithm in minimizing Feedforward Neural Network loss functions. It uses the chain rule to compute iteratively the error of each neuron in a Network, from the output to the input layer.

Errors at the output layer: Let's begin by computing $\delta^L_j, i \in \{1,\ldots,c\}$, errors of neurons in the last layer L. By using the chain rule, we have

$$\begin{aligned}\delta^L_j &= \frac{\partial C_{\mathbf{x}}}{\partial a^L_j}\frac{\partial a^L_j}{\partial z^L_j} \\ &= \left(y_j - a^L_j\right)\varphi'\left(z^L_j\right)\end{aligned} \tag{10}$$

Because the loss function depends on z^L_j, through $\mathbf{a}^L_j$ only.

Errors at any hidden layer: error δ^l_j of any hidden neuron j at any layer l. The weighted input z^l_j of a hidden layer l is linked to the loss function through all weighted inputs $(z^{l+1}_k)_k$ to the next layer.

$$\begin{aligned}\delta^l_j &= \sum_k \frac{\partial C_{\mathbf{x}}}{\partial z^{l+1}_k}\frac{\partial z^{l+1}_k}{\partial z^l_j} \\ &= \sum_k \delta^{l+1}_k \frac{\partial z^{l+1}_k}{\partial z^l_j}.\end{aligned}$$

Using the chain rule, we have

$$\begin{aligned}\frac{\partial z^{l+1}_k}{\partial z^l_j} &= \frac{\partial z^{l+1}_k}{\partial a^l_j}\frac{\partial a^{l+1}_k}{\partial z^l_j} \\ &= w^{l+1}_{jk}\varphi'\left(z^l_j\right).\end{aligned}$$

$$\delta^l_j = \varphi'\left(z^l_j\right)\sum_k w^{l+1}_{jk}\delta^{l+1}_k. \tag{11}$$

The above expression tells us that error functions at any hidden layer are given by the weighted sum of the error functions at the next layer. Which means that errors are computed backwards, hence the name backpropagation. By writing partial derivatives, $\frac{\partial C_{\mathbf{x}}}{\partial w^l_{ij}}$ with respect to δ^l_j, the gradient descent updating rule is rewritten

$$w^l_{ij} \to w^l_{ij} - \frac{\alpha}{m}\sum_{\mathbf{x}\in X_m}\sum_{j=1}^{n_h}\delta^l_j(\mathbf{x})\mathbf{a}^{l-1}_i(\mathbf{x}). \tag{12}$$

Where n_h is the number of neurons in the lth layer. Typically, in DL algorithms, the SGD algorithm is combined with backpropagation, where we have to compute the gradient of a loss function, to be minimized for a large set of data. The implementation of this algorithm is done in a few steps:

1. Provide a set of training examples
2. For each example x:give $\mathbf{a}^1(\mathbf{x})$, and perform the following steps:
   - **Do a Feedforward**: For $l = 2, 3, \ldots, L$ compute $\mathbf{z}^l(\mathbf{x}) = W^l\mathbf{a}^{l-1}(\mathbf{x}) + \mathbf{b}^l$ with $\mathbf{a}^l(\mathbf{x}) = \varphi(\mathbf{z}^l(\mathbf{x}))$.
   - **Output error function** δ^L:Compute $\boldsymbol{\delta}^L(\mathbf{x}) = \nabla C_{\mathbf{x}} \odot \varphi'(\mathbf{z^L}(\mathbf{x}))$.
   - **Backpropagate the error**: For $l = L-1, L-2, \ldots, 2$ compute $\delta^l(\mathbf{x}) = ((W^{l+1})^T\delta^{l+1}(\mathbf{x})) \odot \varphi'(\mathbf{z}^l(\mathbf{x}))$
3. Gradient descent:For $l = L, L-1, \ldots, 2$ update the weights according to the formula $W^l \to W^l - \frac{\alpha}{m}\sum_{\mathbf{x}\in X_m}\delta^l(\mathbf{x})(\mathbf{a}^{l-1}(\mathbf{x}))^T$. We can also show with small computations that the update formula for the vector b^l containing the bias terms in any l layer is written:$b^l \to b^l - \frac{\alpha}{m}\sum_{\mathbf{x}\in X_m}\delta^l(\mathbf{x})$

To implement stochastic gradient descent in practice, an external loop generating mini training example runs, and an external loop running through several training epochs are required. However, these were omitted for simplicity.

4. Regularization techniques

One of the most serious problems in training ML models, particularly for NN, is overfitting. This problem occurs when a training model is too complex.

4.1. L^1 and L^2 regularization techniques

A widely used technique to reduce a model complexity is to add a regularization term [26] to the loss function C. The new model loss function C_λ is defined as follows:

$$C_\lambda = C + \lambda\Omega(W)$$

These, update the general cost function by adding another term known as the regularization term, where Ω is L^1 or L^2 norm and w is the NN weight parameters.

4.1.1. L^2 regularization

The L^2 regularization term, commonly known as **weight decay**. The idea of this technique also known as **ridge regression** or **Tikhonov** regularization [78], is to add a L^2 term to the function to be minimized, in this case $\Omega(W) = \frac{1}{2}\|W\|^2_2$. This added term in L^2 norm imposes the weights to live in a sphere of radius inversely proportional to the regularization parameter [[26], p. 249] λ. In this context, the updating rule, using gradient descent strategy becomes

$$w^l_{ij} \to \left(1 - \frac{\alpha\lambda}{n}\right)w^l_{ij} - \frac{\alpha}{n}\sum_{i=1}^{n}\frac{\partial C_{\mathbf{x}_i}}{\partial w^l_{ij}}. \tag{13}$$

this means that, after each iteration, the weights are multiplied by a factor slightly smaller 1. It tends to force the model to prefer small weights.

4.1.2. L^1 regularization

L^1 regularization modifies the loss function by adding a L^1 term, i.e. $\Omega(W) = \sum_{w\in W}|w|$. The idea behind this technique is to regularize the loss function by removing the irrelevant features from the training model. In this situation, the updating rule is written

$$w_{ij}^l \to w_{ij}^l - \frac{\alpha\lambda}{n} sgn\left(w_{ij}^l\right) - \frac{\alpha}{n}\sum_{i=1}^{n}\frac{\partial C_{\mathbf{x}_i}}{\partial w_{ij}^l}. \tag{14}$$

Where $sgn(w_{ij}^l)$ is the sign of w_{ij}^l. Both types of regularization try to penalize the big weights when it's necessary by shrinking them after each updating step, but the way of shrinkage is different [26]. When L^2 regularization is used, the weights are shrunk by an amount proportional to w_{ij}^l, whereas in L^1 regularization, the weights are shrunk by constant quantity toward to zero. As shown in Fig. 5 (graph on the left), in a two dimensional space, the L^1 norm defines a parameter space bounded by a parallelogram at the origin. In this case, the loss function is likely to hit the vertices of the parallelogram rather than its edges. L^1 regularization removes some of the parameters, thus L^1 technique can be used as a feature selection technique. On the other hand, the L^2 regularization defines a circle whose radius size is inversely proportional to the regularization parameter (see Fig. 6).

4.2. Dropout technique

In training a NN, Dropout technique regularizes learning by dropping out some hidden units with certain probability. This is equivalent to modifying [72] the NN by setting some hidden activation functions to zero. Using Dropout, we can formally define the NN as follows:

$$\tilde{a}_j^l = \delta_j^l a_j^l, \tag{15}$$

At each neuron j, in a hidden layer l, the output activation a_j^l is multiplied by a sampled variable δ_j^l, to produce thinned output activations $\tilde{a}_j^l$. These thinned functions are then used as inputs to the next layer and the same process is applied at each layer. This application is equivalent to sampling a sub Neural Networks from a larger network. Where δ_j^l is a Bernoulli random variable ($\delta_j^l \hookrightarrow$ Bernoulli(p^l)) of parameter p^l, i.e. a neuron in the lth layer is kept with a probability of p^l and removed with a probability $1 - p^l$.

Srivastava et al. [72] suggested that, applying Dropout to a NN with n units can be seen as sampling 2^n sub Networks with weight sharing. In the test phase, as it is not always practical to take the mean of 2^n models, an approximate averaging method is used. The idea is to approximate the exponentially many Networks by a single NN without Dropout. To correct the fact that training outgoing weights of a layer are obtained under condition that neurons were retained with a probability p, the weights are simply multiplied by p. This approximation has been proved for logistic and linear regression models [72,79]. But, for Deep Neural NetworksDNNs, there is an unknown gap between the expected output of exponential sub Networks and the output of a single deterministic model. Ma et al. [46] showed that under some assumptions on input data, the gap is controlled and it can be used to regularize the single NN.

In this work, without any assumption of input data, we quantify explicitly the gap and then show how it related to L^2 regularization.

4.2.1. Dropout application to linear networks

To see more clearly the relationship between L^2-regularization, we start by studying the problem in a very simple case, where all activation functions in the model are linear. Consider a NN, where all units are linear (i.e. $\mathbf{a}^l = \mathbf{a}^{l-1}W^l$, where $\mathbf{a}^l$ and W^l are the output vectors and weight matrix of layer $l \in \{1,\dots,L\}$ respectively). The Dropout NN loss function is

$$\frac{1}{n}\sum_{\mathbf{x}\in X}\left\|\mathbf{y}(\mathbf{x}) - \mathbf{a}^{L-1}(\mathbf{x})\tilde{W}^L\right\|_2^2 + \frac{1-p^{L-1}}{p^{L-1}}\left\|\Sigma^{L-1}\tilde{W}^L\right\|_2^2. \tag{16}$$

$\mathbf{y}(\mathbf{x})$ is the output vector given an input vector $\mathbf{x}$. $\Sigma^{L-1} = \left(\frac{1}{n} diag(\mathbf{a}^{L-1}(X)(\mathbf{a}^{L-1}(X))^T(X))\right)^{\frac{1}{2}}$, $\tilde{W}^L = p^{L-1}W^L$. Given a matrix A, we denote by $diag(A)$, a diagonal matrix with the same size and diagonal elements as A.

At each layer l, we define a matrix $\mathbf{a}^l(X)$ whose columns correspond to the values taken by the vector of the activation function $\mathbf{a}^l$ across input data: $\mathbf{a}^l(X) = (a_i^l(\mathbf{x}_j))$, $1 \le i \le m$, $1 \le j \le n$, where $a_i^l(\mathbf{x}_j)$ is the ith output neuron in (l)th layer of the jth input and m is the number of neurons in the layer.

Proof. Training a standard nn without dropping neurons is done by minimizing the following loss function:

$$\frac{1}{n}\sum_{\mathbf{x}\in X}\|\mathbf{y}(\mathbf{x}) - \mathbf{a}^{L-1}(\mathbf{x})W^L\|_2^2 \tag{17}$$

Dropout modifies the training process and the loss function in (17) becomes

$$\frac{1}{n}\sum_{\mathbf{x}\in X}\mathbb{E}_{\delta^{L-1}}\|\mathbf{y}(\mathbf{x}) - (\delta^{L-1}(\mathbf{x})\odot\mathbf{a}^{L-1}(\mathbf{x}))W^L\|_2^2. \tag{18}$$

Where δ^{L-1} is a random vector of the layer $L-1$ with $\delta_i^{L-1} \hookrightarrow Bernoulli(p^{L-1})$ and $\odot$ denotes the Hadamard product. Using the formula $E(X^2) = (E(X))^2 + Var(X)$ for a random variable X, we show that (18) is equal to

$$\frac{1}{n}\sum_{\mathbf{x}\in X}\|\mathbf{y}(\mathbf{x}) - \mathbb{E}_{\delta^{L-1}}\left(\left(\delta^{L-1}(\mathbf{x})\odot\mathbf{a}^{L-1}(\mathbf{x})\right)W^L\right)\|_2^2 + \frac{1}{n}\sum_{\mathbf{x}\in X}Var\left(\left(\delta^{L-1}(\mathbf{x})\odot\mathbf{a}^{L-1}(\mathbf{x})\right)W^L = \right.$$

$$\frac{1}{n}\sum_{\mathbf{x}\in X}\|\mathbf{y}(\mathbf{x}) - p^{L-1}\mathbf{a}^{L-1}(\mathbf{x})W^L\|_2^2 + \frac{1}{n}\sum_{\mathbf{x}\in X}W^L Var\left(\delta^{L-1}(\mathbf{x})\odot\mathbf{a}^{L-1}(\mathbf{x})\right)(W^L)^T.$$

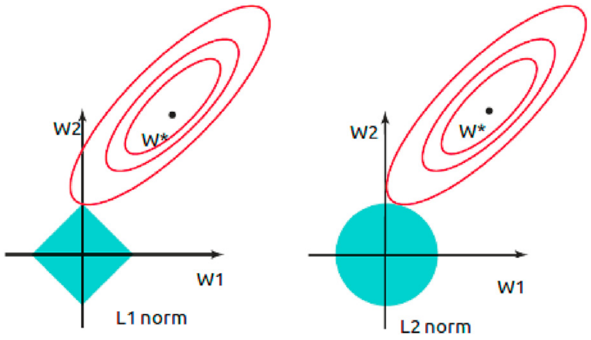


Fig. 5. Two dimensional graphical interpretation of L^1 and L^2 regularizations.

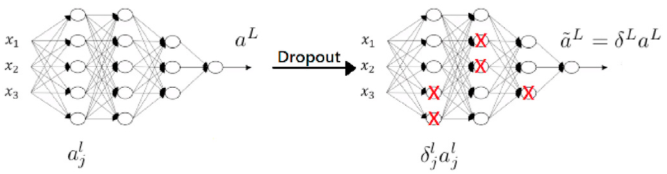


Fig. 6. Dropout.

As $Var(\delta^{L-1}(\mathbf{x}) \odot \mathbf{a}^{L-1}(\mathbf{x})) = \left(\frac{1-p^{L-1}}{p^{L-1}}\right)\mathbf{a}^{L-1}(\mathbf{x})\mathbf{a}^{L-1}(\mathbf{x})^T$, we obtain the desired result. Under the assumption that input layers follow a Gaussian distribution with standard deviation σ, Dropout is equivalent in expectation to L^2-regularization. The regularization parameter λ is a function of $\frac{1-p^{L-1}}{p^{L-1}}\sigma^2$ which increases (resp. decreases) with the variance of input layers σ^2 (resp. with p^{L-1}). Thus, Dropout regularization consists of detecting the inputs with more variance and shrink their weights.

4.2.2. Dropout application to non linear networks

Here, we try to generalize the relationship between Dropout and L^2-regularization to Networks with nonlinear units. Consider a NN with a non linear activation function, i.e., $\mathbf{a}^l = \varphi(\mathbf{a}^{l-1}W^L)$. Dropout training expected loss function is given by

$$\frac{1}{n}\sum_{\mathbf{x}\in X}\left\|\mathbf{y}(\mathbf{x}) - \varphi\left(\left(\tilde{W}^L\right)^T\mathbf{a}^{L-1}(\mathbf{x})\right)\right\|_2^2 + \frac{1}{2n}\left(\frac{1-p}{p}\right)\sum_{\mathbf{x}\in X}\|\varphi''\left(\mathbf{a}^{L-1}(\mathbf{x})\tilde{W}^L\right)\Sigma_{\mathbf{x}}^{L-1}\tilde{W}^L\|_2^2. \tag{19}$$

Where $\Sigma_{\mathbf{x}}^{L-1} = \left(\mathbf{a}^{L-1}(\mathbf{x})(\mathbf{a}^{L-1}(\mathbf{x}))^T\right)^{\frac{1}{2}}$ and $\tilde{W}^L = p^{L-1}W^L$.

Proof. We know that a non linear Dropout Network training loss is defined as

$$\frac{1}{n}\sum_{\mathbf{x}\in X}\mathbb{E}_{\delta^{L-1}}\|\mathbf{y}(\mathbf{x}) - \varphi((\delta^{L-1}(\mathbf{x})\odot\mathbf{a}^{L-1}(\mathbf{x}))W^L)\|_2^2. \tag{20}$$

Using triangle inequality, (20) is bounded by

$$\frac{1}{n}\sum_{\mathbf{x}\in X}\|\mathbf{y}(\mathbf{x}) - \varphi(p^{L-1}\mathbf{a}^{L-1}(\mathbf{x})W^L)\|_2^2 + \frac{1}{n}\sum_{\mathbf{x}\in X}\|\mathbb{E}_{\delta^{L-1}(\mathbf{x})}(\varphi((\mathbf{a}^{L-1}(\mathbf{x})\odot\delta^{L-1}(\mathbf{x}))W^L)) - \varphi(p^{L-1}\mathbf{a}^{L-1}(\mathbf{x})W^L)\|_2^2.$$

Now, by applying a second order Taylor expansion of φ around $\mathbb{E}_{\delta^{L-1}}((\mathbf{a}^{L-1}\odot\delta^{L-1}(\mathbf{x})W^L)) = p^{L-1}\mathbf{a}^{L-1}(\mathbf{x})W^L$ and by posing $Z = (\mathbf{a}^{L-1}(\mathbf{x})\odot\delta^{L-1}(\mathbf{x}))W^L - p^{L-1}\mathbf{a}^{L-1}(\mathbf{x})W^L$, we have $\varphi((\mathbf{a}^{L-1}(\mathbf{x})\odot\delta^{L-1}(\mathbf{x}))W^L) = \varphi(p^{L-1}\mathbf{a}^{L-1}(\mathbf{x})W^L) + \varphi'(p^{L-1}\mathbf{a}^{L-1}(\mathbf{x})W^L)Z + \frac{1}{2}\varphi''(p^{L-1}\mathbf{a}^{L-1}(\mathbf{x})W^L)ZZ^T$.

Then $\mathbb{E}_{\delta^{L-1}(\mathbf{x})}(\varphi((\mathbf{a}^{L-1}(\mathbf{x})\odot\delta^{L-1}(\mathbf{x}))W^L)) - \varphi(p^{L-1}\mathbf{a}^{L-1}(\mathbf{x})W^L) = \frac{1}{2}\varphi''(p^{L-1}\mathbf{a}^{L-1}(\mathbf{x})W^L)Var(ZZ^T)$. Because Z is centered i.e., $\mathbb{E}_{\delta^{L-1}(\mathbf{x})}(Z) = 0$. Thus, an upper bound of (20) is given by

$$\frac{1}{n}\sum_{\mathbf{x}\in X}\left\|\mathbf{y}(\mathbf{x}) - \varphi\left(\mathbf{a}^{L-1}(\mathbf{x})\tilde{W}^L\right)\right\|_2^2 + \frac{1}{2n}\left(\frac{1-p^{L-1}}{p^{L-1}}\right)\sum_{\mathbf{x}\in X}\|\varphi''\left(\mathbf{a}^{L-1}(\mathbf{x})\tilde{W}^L\right)\Sigma_{\mathbf{x}}\tilde{W}^L\|_2^2.$$

Here again, Dropout can be seen as regularizer, where the regularizer represents the gap between $\mathbb{E}_{\delta^{L-1}}(\varphi((\mathbf{a}^{L-1}\odot\delta^{L-1})W^L))$ the expected output of exponential thinned Networks produced by applying Dropout and $\varphi(p^{L-1}\mathbf{a}^{L-1}W^L)$, the output of a single deterministic Network, in which the weights are scaled by p^{L-1} to compensate the fact that they are learned under conditions in which $1-p^{L-1}$ of hidden units where dropped out. In this case, Dropout training model can be seen as an L^2-regularization where, the regularizer λ depends on: Dropout rate; the variance of each input and output layer.

4.2.3. Dropout with others regularization techniques

Dropout is known to improve training model performance when it is combined with other regularization techniques. Batch normalization, introduced by Ref. [35], is a regularization technique used to speed up the training and improve performance of Deep NNS. In the training of a DNN, the distribution of each layer's inputs change, as the parameters of all layers that come before it, variate. This can slow down by requiring small learning rates and careful parameter initialization. Given a batch of sample used to update parameters, batch normalization normalizes the inputs of each layer by recentering and rescaling (subtracting the mean and dividing by the batch standard deviation). Thus, batch normalization prevents layers inputs to have large standard deviations [35]. show experimentally that batch normalization with large learning rate, speed up significantly training as it can eliminate the need for Dropout. In fact, as discussed in Section ??, Dropout look for layer's inputs with more varitions and shrink their weights, this function of shrinking is then largely reduced by batch normalization application. Combining dropout with batch normalization [25,35,43] can improve DNNS prediction accuracy. Dropout regularization is known to give a significant improvement when it is combined with others regularization methods such as max-norm [72] and weight normalization [67]. Rather than constraining whole weight matrix of each layer as in L^2 regularization, these constrain each column of the weight matrix to prevent separately any hidden neuron from having very large weights. Max-norm regularization consists to constrain the incoming weight vector of each hidden neuron to live in a ball of radius c, where c is a hyper-parameter. Weight normalization constrains incoming weight vectors to have unit norm.

5. Materials and methods

All models in this work are constructed using Keras and Tensorflow open source libraries [38]. Logistic regression is built using a Feedforward classification network without hidden layers, this model can be extended later to a more complex classification Network, depending on

Table 1
Different datasets used in this study.

Dataset	#of samples	Class 1	Class 2	# of SNPs
Breast-Kidney	604	344	260	10937
Colon-Kidney	546	286	260	10937
Breast-Colon	630	344	286	10937
Colon-Prostate	286	69	355	10937

the problem complexity. Stochastic gradient descent is adopted in all experiments as an optimization strategy. Two types of datasets are included in our experiments, Expression Project for Oncology (expO) cancer datasets and 1000 Genomes Project ethnicity datasets respectively used for training logistic regression and FFN models.

Individuals in selected datasets are humans that are represented by the list of their SNPs. Each SNP is represented by its genotype (i.e. genetic information) at a specific locus. In a diploid organism at each locus, there are two copies of alleles, one comes from the father and other from the mother. Consequently, a genotype takes one of three values for a diploid organism: 0 (homozygous reference), 1 (heterozygous) and 2 (homozygous alternate). The homozygous reference refers to the base that is found in the reference genome, an homozygous alternate refers to any base, other than the reference, that is found at that locus and genotype is said heterozygous at a given position, when the two alleles are different.

The input of a model is a matrix X of size $n \times d$, where n is the number of individuals included in the study and d correspond to the number of features (SNPs). The output y takes discrete value(s) between 0 and 1.

5.1. Expression Project for Oncology(expO) cancer datasets

The different cancer samples included in this study (see Table 1), are downloaded from Ref. [11]. The original datasets can be obtained from the Expression Project for Oncology (expO) that was deposited at Gene Expression Omnibus (GEO) repository [7], with accession number GSE2109. The objective of expO is to obtain and perform gene expression analysis on cancer tissue samples and assemble the patient's long term clinical results.

5.2. 1000 Genomes Project dataset

The 1000 Genomes Project [16] took advantage of developments in Next-generation sequencing (NGS), which allows to sequence DNA and RNA much more quickly and cheaply. It's the first project to sequence the genomes of a large number of people in populations from different regions and countries. In this study, $n = 3450$ is the number of individuals sampled worldwide from 26 populations and $d = 315345$ is the number of SNPs. The desired output of the model is a vector $Y \in \mathbb{R}^c$, whose components correspond to the 26 classes of populations (i.e. $c = 26$). The model consists of an input layer, an output layer and two hidden layers of equal size. Given the input matrix X, the model output is a vector $\mathbf{a}^3 \in \mathbb{R}^c$. A reluaction function is used in the two hidden layers followed by a sotmax layer to perform ancestry prediction.

6. Experiments

In this section, we present the effects of regularization techniques on training models for different datasets (see Table 1).

6.1. Cancer dataset classification using logistic regression

Un-regularized logistic regression results are reported in Table 2, despite its simplicity, logistic regression model gives good classification accuracy on these cancer datasets. To improve prediction capacities of the present model, a penalty term is added and obtained results are presented in Table 3 and Table 4 for L^1 and L^2 regularization added term, respectively. We can observe from theses table that penalization with the appropriate regularization parameter improves the classification accuracy. For example, when L^1 regularization is used with regularizer $\lambda = 10^{-3}$, the classification accuracy on Breast-Kidney dataset goes from 96.53 to **99.01**. Similarly, when L^2 penalization is applied($\lambda = 10^{-3}$), the prediction accuracy on Breast-Colon dataset increases from 94.44 in unregularized case to **99.44**.

Table 2
Unregularized logistic reg.

Dataset	Accuracy (in %)
Breast-Kidney	96.53
Colon-Kidney	97.82
Breast-Colon	94.13
Colon-Prostate	97.46

Table 3
Logistic reg. with L^1 norm.

Dataset	Regularization L^1	Accuracy(in %)
Breast-Kidney	$\lambda = 10^{-2}$	97.36
	$\lambda = 10^{-3}$	**99.01**
Colon-Kidney	$\lambda = 10^{-2}$	95.82
	$\lambda = 10^{-3}$	97.45
Breast-Colon	$\lambda = 10^{-2}$	94.44
	10^{-3}	93.17
Colon-Prostate	$\lambda = 10^{-2}$	**98.59**
	$\lambda = 10^{-3}$	98.03

Table 4
Logistic reg. with L^2 norm.

Dataset	Regularization L^2	Accuracy(in %)
Breast-Kidney	$\lambda = 10^{-2}$	98.18
	$\lambda = 10^{-3}$	98.02
Colon-Kidney	$\lambda = 10^{-2}$	98.18
	$\lambda = 10^{-3}$	**98.91**
Breast-Colon	$\lambda = 10^{-2}$	92.86
	$\lambda = 10^{-3}$	**99.44**
Colon-Prostate	$\lambda = 10^{-2}$	97.18
	$\lambda = 10^{-3}$	96.62

Table 5
MLP accuracy vs its size.

# of units by hidden layer	Accuracy(in %)
[50]	81.33
[50-50]	81.68
[100]	90.68
[100−100]	92.70
[100-100-100]	90.49
[500-500-500]	90.46

6.2. Ancestry prediction using a multilayer perceptron (MLP)

In this subsection, FNN is used on 1000 Genome Project ethnicity dataset to predict individuals ancestries. As in the preceding subsection, we start with a simple logistic regression model, which gives a low prediction accuracy of **54.64%**. To achieve better prediction results, a MLP with equal hidden units is constructed and the obtained results by varying the model complexity are reported in Table 5. As expected, the model prediction accuracy starts by increasing with its complexity until some level (two hidden layers with 100 units for each one), then it starts to drop. Because beyond this stage, the training model is considered too complex and it overfits.

6.2.1. Classification with autoencoder

We start by using a classification Network with one hidden layer of 50 units with reconstruction path. This gives an accuracy of 84.85%. When another hidden layer of 50 neurons is added between hidden the representation $\mathbf{a}_h$ and the output layer $\mathbf{a}^L$, as described in Fig. 4(where MPL=[50]). This last gives an accuracy of 85.36%. Training the classification Network with a reconstruction path is difficult due to the high dimensionality of the input data.

Table 6
Prediction accuracy for Dropout, batch normalization and dropout combination with batch normalization.

# of uni. by hid. layer	Drop.(p=0.2)	Drop.(p=0.5)	Bat.norm	Drop.(p=0.2)+Bat. norm	Drop.(p=0.5)+Bat. norm
	(accuracy in%)	(accuracy in%)	(accuracy in%)	(accuracy in%)	(accuracy in%)
[50]	90.32	87.54	90.58	91.48	92.61
[50-50]	89.94	40.96	91.01	92.58	92.93
[100]	88.81	92.26	90.75	91.65	93.01
[100−100]	90.32	64.12	89.19	92.46	93.00

Table 7
Prediction accuracy for L^1, L^2, Dropout regularization techniques and Dropout combination with others regularization techniques.

# of uni. by hid. layer	$L^1(\lambda = 10^{-4})$ reg.	$L^2(\lambda = 10^{-3})$ reg.	Drop.(p=0.5)+Bat. norm	Drop.(p=0.5)+Ma.norm	Drop.(p=0.2)+Un.norm
	(accuracy in%)	(accuracy in%)	(accuracy in%)	(accuracy in%)	(accuracy in%)
[50]	92.13	92.61	92.61	92.35	93.25
[50-50]	91.77	90.75	92.93	83.68	90.32
[100]	92.70	92.70	93.01	93.83	**94.43**
[100−100]	92.29	93.39	93.00	90.17	91.94
[100-100-100]	90.87	91.01	92.42	89.68	92.19

6.2.2. *Classification with regularization*

When Dropout is used alone, the choice of its rate p is very important as in Table 6, we have to be more careful about it. Combining Dropout with batch normalization improves the performance training model and makes the choice of p less important. L^1 and L^2 regularization, give good prediction accuracy and outperform Dropout as observed in Table 7. However, when Dropout is combined with batch normalization, max-norm and unit-norm outperforms the traditional regularization techniques. We have obtained our best prediction accuracy **94.43**%, when Dropout is combined with unit norm constraint.

7. Discussion

Regularized Logistic Regression has achieved good results compared to previous machine learning approaches tested on the same cancer samples [70,73]. For instance, Stiglic et al. [73] combined different feature selection methods such as Vector Machines Recursive Feature Elimination (SVM-RFE) and ReliefF followed by *SVM* or k-nearest neighbors. To the best of our knowledge, the best results on these datasets were reported in Ref. [70], where the authors used Stacked Sparse Autoencoders (SSAE) to select most relevant features followed by a classification Neural Network to categorize the samples. Despite its simplicity, the proposed approach outperforms SSAE on datasets such as Breast-Kidney and Breast-Colon (see Table 8). In Stacked Sparse Autoencoders [36,45] many Autoencoder layers are stacked together to form an unsupervised learning algorithm, where the encoder layer computed by an Autoencoder will be used as the input to another Autoencoder layer. In practice, a logistic regression model may be better than a NN for relatively small data sets and simple classification tasks, where the classes are more or less linearly separable. Indeed, the latter are more difficult to train, require more training samples and are more prone to overfitting than logistic regression.

Logistic regression application to anscentry prediction dataset lead to poor prediction accuracy, which is due to the large dimension of input features and high nonlinear correlation between them, and to the genetic similarity between some ethnic of groups of populations.

Training a NN with an Autoencoder reconstruction path improved the results. However, training an Autoencoder in conjunction with the classification Network makes the high dimensional optimization problem more difficult to solve than simply training the classification Network, yielding in a higher classification error. To improve the results, regularization techniques are used. One can notice that traditional regularization technique's application has more improved the prediction accuracy of the model compared to Dropout. This could be attributed to the fact that Dropout is less effective than L^1 and L^2 regularization techniques [58] when the training model is not complex (as for our model). Combining Dropout with techniques such as Batch normation, Unit norm constraint or Max norm enabled the training model to achieve its best accuracy. The obtained results are compared to the results in Table 9 obtained by Ref. [64] on the same dataset. In Ref. [64], the authors proposed auxiliary NNs to predict the parameters of the first hidden layer of the classification NNs and different features embedding techniques such as Random projection(RP), Per class histogram, and SNPtoVec have also been proposed. Achieving such prediction accuracy obtained with SNP data, these regularization techniques will allow us to face more complicated problems in many domains such as preventive medicine.

8. Conclusion

In this work, we have explained stochastic gradient descent optimization technique with back-propagation in training DL algorithms. To prevent overfitting problem, regularization techniques are studied and, theoretical relationship between Dropout and L^2 regularization is established. Experimental results have shown that Dropout, when it is combined with techniques such as batch normalization, max-norm or unit-norm gives better performance than L^1 and L^2 regularization techniques.

For future work, we expect to further study these regularization techniques in DNN and use them to analyze gene expression profile data with the aim of predicting rare diseases.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This project was partly funded by H3ABioNet, which is supported by the National Institutes of Health Common Fund under grant number U41HG006941.

Appendix

In this section, we report results obtained by Singh et al. [70](Table 8) and those obtained by Romero et al. [64](Table 9).

Table 8
Logistic reg.vs SSAE.

Dataset	Stacked Sparse Autoencoders	Logistic regression
Breast-Kidney	98.4	99.01
Colon-Kidney	99.5	98.91
Breast-Colon	97.3	99.44
Colon-Prostate	99.7	98.59

Table 9
Reported results in Ref. [64].

Model & Embedding	Mean Misclassif. Error. (%)	# of free param.
Basic	8.31 ± 1.83	31.5 M
Raw end2end	8.88 ± 1.41	21.27K
Random Projection	9.03 ± 1.20	10.1K
SNP2Vec	7.60 ± 1.28	10.1K
Per class histograms	7.88 ± 1.40	7.9K
Basic with reconstruction	7.76 ± 1.38	63 M
Raw end2end with reconstruction	8.28 ± 1.92	227.3K
Random Projection with reconstruction	8.03 ± 1.0.3	20.2K
SNP2Vec with reconstruction	7.88 ± 0.72	20.2K
Per class histograms with reconstruction	7.44 ± 0.45	15.8K

References

[1] Akhtar N, Mian A. Threat of adversarial attacks on deep learning in computer vision: a survey. IEEE Access 2018;6:14410–30. https://doi.org/10.1109/ACCESS.2018.2807385.

[2] Almagro Armenteros JJ, Sønderby CK, Sønderby SK, Nielsen H, Winther O. Deeploc: prediction of protein subcellular localization using deep learning. Bioinformatics 2017;33:3387–95. https://doi.org/10.1093/bioinformatics/btx431.

[3] Amari S. Backpropagation and stochastic gradient descent method. Neurocomputing 1993;5:185–96. https://doi.org/10.1016/0925-2312(93)90006-O.

[4] Amodei D, Ananthanarayanan S, Anubhai R, Bai J, Battenberg E, Case C, Casper J, Catanzaro B, Cheng Q, Chen G, et al. Deep speech 2: end-to-end speech recognition in English and Mandarin. In: Inter. conf. on ML; 2016. p. 173–82. https://arxiv.org/abs/1512.02595.

[5] Bacchi S, Oakden-Rayner L, Zerner T, Kleinig T, Patel S, Jannes J. Deep learning natural language processing successfully predicts the cerebrovascular cause of transient ischemic attack-like presentations. Stroke 2019;50:758–60. https://doi.org/10.1161/STROKEAHA.118.024124.

[6] Baldi P, Sadowski PJ. Understanding dropout. In: NIPS; 2013. p. 2814–22. https://doi.org/10.1016/j.artint.2014.02.004.

[7] Barrett T, Edgar R. [19] gene expression omnibus: microarray data storage, submission, retrieval, and analysis. Meths. in enzy. 2006;411:352–69. doi: S0076687906110198.

[8] Bilen M, Işik AH, Yiğit T. A hybrid artificial neural network-genetic algorithm approach for classification of microarray data. In: 2015 23nd SPCA conf. (SIU), IEEE; 2015. p. 339–42. https://doi.org/10.1109/SIU.2015.7129828.

[9] Bottou L. Stochastic gradient learning in neural networks. Proceedings of Neuro-Nımes 1991;91:12.

[10] Bottou L. Stochastic gradient descent tricks. In: Neural networks: tricks of the trade. Springer; 2012. p. 421–36.

[11] Cancer D. Openml; 2015. https://www.openml.org/search?type=data. [Accessed 7 September 2020].

[12] Chatterjee S, Hadi AS. Sensitivity analysis in linear regression, vol. 327. John Wiley & Sons; 2009.

[13] Chen Y, Li Y, Narayan R, Subramanian A, Xie X. Gene expression inference with deep learning. Bioinformatics 2016;32:1832–9. https://doi.org/10.1093/bioinformatics/btw074.

[14] Collins FS, Brooks LD, Chakravarti A. A dna polymorphism discovery resource for research on human genetic variation. Geno. research 1998;8:1229–31. https://doi.org/10.1101/gr.8.12.1229.

[15] Collobert R, Weston J. A unified architecture for natural language processing: deep neural networks with multitask learning. In: Proceed. of the 25th inter. conf. on ML; 2008. p. 160–7. https://doi.org/10.1145/1390156.1390177.

[16] Consortium GP, et al. A map of human genome variation from population-scale sequencing. Nature 2010;467:1061. https://doi.org/10.1038/nature09534.

[17] Curry HB. The method of steepest descent for non-linear minimization problems. Quart. of App. Maths. 1944;2:258–61. https://www.ams.org/journals/qam/1944-02-03/S0033-569X-1944-10667-3/S0033-569X-1944-10667-3.pdf.

[18] Denoeux T. Logistic regression, neural networks and dempster–shafer theory: a new perspective. Knowl Base Syst 2019;176:54–67.

[19] Dreiseitl S, Ohno-Machado L. Logistic regression and artificial neural network classification models: a methodology review. J Biomed Inf 2002;35:352–9.

[20] Fakoor R, Ladhak F, Nazi Z, Huber M. Using deep learning to enhance cancer diagnosis and classification. In: Proceed. of the inter. conf. on ML. New York, USA: ACM; 2013. https://doi.org/10.1109/ICSCAN.2018.8541142.

[21] Fort G, Lambert-Lacroix S. Classification using partial least squares with penalized logistic regression. Bioinformatics 2005;21:1104–11. https://doi.org/10.1093/bioinformatics/bti114.

[22] Fu X, Wei Y, Xu F, Wang T, Lu Y, Li J, Huang JZ. Semi-supervised aspect-level sentiment classification model based on variational autoencoder. Knowl Base Syst 2019;171:81–92.

[23] Gal Y, Ghahramani Z. Dropout as a bayesian approximation: representing model uncertainty in deep learning. In: International conference on machine learning. PMLR; 2016. p. 1050–9.

[24] Ganesan N, Venkatesh K, Rama MA, Palani AM. Application of neural networks in diagnosing cancer disease using demographic data. Int J Chem Appl 2010;1:76–85. https://doi.org/10.5120/476-783.

[25] Garbin C, Zhu X, Marques O. Dropout vs. batch normalization: an empirical study of their impact to deep learning. Multimed Tool Appl 2020:1–39. https://doi.org/10.1007/s11042-019-08453-9.

[26] Goodfellow I, Bengio Y, Courville A, Bengio Y. Deep learning, vol. 1. MIT press Cambridge; 2016. https://doi.org/10.1007/s10710-017-9314-z.

[27] Group, I.S.M.W., et al.. A map of human genome sequence variation containing 1.42 million single nucleotide polymorphisms. Nature 2001;409:928. https://doi.org/10.1038/35057149.

[28] Guo J, He H, He T, Lausen L, Li M, Lin H, Shi X, Wang C, Xie J, Zha S, et al. Gluoncv and gluonnlp: deep learning in computer vision and natural language processing. J Mach Learn Res 2020;21:1–7. 1907.04433.

[29] Hannun A, Case C, Casper J, Catanzaro B, Diamos G, Elsen E, Prenger R, Satheesh S, Sengupta S, Coates A, et al. Deep speech: scaling up end-to-end speech recognition. arXiv preprint arXiv, 1412.5567; 2014. https://arxiv.org/abs/1412.5567.

[30] Hecht-Nielsen R. Theory of the backpropagation neural network. In: N. netw. for percep. Elsevier; 1992. p. 65–93. https://doi.org/10.1016/B978-0-12-741252-8.50010-8.

[31] Helmbold DP, Long PM. Surprising properties of dropout in deep networks. In: Conference on learning theory. PMLR; 2017. p. 1123–46.

[32] Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. science 2006;313:504–7. https://doi.org/10.1126/science.1127647.

[33] Hinton GE, Srivastava N, Krizhevsky A, Sutskever I, Salakhutdinov RR. Improving neural networks by preventing co-adaptation of feature detectors. 2012. arXiv preprint arXiv, 1207.0580. http://arxiv.org/abs/1207.0580.

[34] Huang K, Hussain A, Wang QF, Zhang R. Deep learning: fundamentals, theory and applications, vol. 2. Springer; 2019.

[35] Ioffe S, Szegedy C. Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv, 1502.03167; 2015.

[36] Katuwal R, Suganthan PN. Stacked autoencoder based deep random vector functional link neural network for classification. Appl Soft Comput 2019;85:105854.

[37] Kendall A, Gal Y. What uncertainties do we need in bayesian deep learning for computer vision?. In: NIPS; 2017. p. 5574–84. https://arxiv.org/pdf/1703.04977.pdf.

[38] Keras T. Keras. 2015. https://www.tensorflow.org/guide/keras/overview. [Accessed 7 September 2020].

[39] Kleinbaum DG, Dietz K, Gail M, Klein M, Klein M. Logistic regression. Springer; 2002.

[40] Konečnỳ J, Richtárik P. Semi-stochastic gradient descent methods. 2013. arXiv preprint arXiv, 1312.1666.

[41] Le L, Patterson A, White M. Supervised autoencoders: improving generalization performance with unsupervised regularizers. Adv Neural Inf Process Syst 2018;31:107–17.

[42] Li F, Zurada JM, Liu Y, Wu W. Input layer regularization of multilayer feedforward neural networks. IEEE Access 2017;5:10979–85.

[43] Li X, Chen S, Hu X, Yang J. Understanding the disharmony between dropout and batch normalization by variance shift. In: Proceed. Of the IEEE conf. On CVPR; 2019. p. 2682–90. https://doi.org/10.1109/CVPR.2019.00279.

[44] Liao JG, Chin KV. Logistic regression for disease classification using microarray data: model selection in a large p and small n case. Bioinformatics 2007;23:1945–51. https://doi.org/10.1093/bioinformatics/btm287.

[45] Liu G, Bao H, Han B. A stacked autoencoder-based deep neural network for achieving gearbox fault diagnosis. Mathematical Problems in Engineering 2018; 2018.

[46] Ma X, Gao Y, Hu Z, Yu Y, Deng Z, Hovy E. Dropout with expectation-linear regularization. 2016. arXiv preprint arXiv, 1609.08017. https://arxiv.org/abs/1609.08017.

[47] Manning CD, Surdeanu M, Bauer J, Finkel JR, Bethard S, McClosky D. The stanford corenlp natural language processing toolkit. In: Proceed. of 52nd ann. meet. of ACL: system demonstrations; 2014. p. 55–60. https://doi.org/10.3115/v1/p14-5010.

[48] Maurya S, Singh V, Dixit S, Verma NK, Salour A, Liu J. Fusion of low-level features with stacked autoencoder for condition based monitoring of machines. In: 2018 IEEE international conference on prognostics and health management (ICPHM), IEEE; 2018. p. 1–8.

[49] Mianjy P, Arora R. On dropout and nuclear norm regularization. In: International conference on machine learning. PMLR; 2019. p. 4575–84. https://arxiv.org/abs/1905.11887.

[50] Min S, Lee B, Yoon S. Deep learning in bioinformatics. Brief. in bioinfo. 2017;18:851–69. https://doi.org/10.1093/bib/bbw068.

[51] Montgomery DC, Peck EA, Vining GG. Introduction to linear regression analysis. John Wiley & Sons; 2021.
[52] Ng A, et al. Sparse autoencoder. CS294A Lect. notes 72. 2011. p. 1–19. http://ailab.chonbuk.ac.kr/seminar_board/pds1_files/sparseAutoencoder.pdf.
[53] Owen AB. A robust hybrid of lasso and ridge regression. Contemp Math 2007;443:59–72. https://doi.org/10.1090/conm/443/08555.
[54] Ozanich E, Gerstoft P, Niu H. A feedforward neural network for direction-of-arrival estimation. J Acoust Soc Am 2020;147:2035–48.
[55] Pal A, Lane C, Vidal R, Haeffele BD. On the regularization properties of structured dropout. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition; 2020. p. 7671–9. https://arxiv.org/abs/1910.14186.
[56] Patel P, Thakkar A. The upsurge of deep learning for computer vision applications. Int J Electr Comput Eng 2020;10:538. https://doi.org/10.11591/ijece.v10i1.pp538-548.
[57] Pei J, Wang W, Osman MK, Gan X. Multiparameter optimization for the nonlinear performance improvement of centrifugal pumps using a multilayer neural network. J Mech Sci Technol 2019;33:2681–91.
[58] Phaisangittisagul E. An analysis of the regularization between l2 and dropout in single hidden layer neural network. In: 2016 7th international conference on intelligent systems, modelling and simulation (ISMS). IEEE; 2016. p. 174–9.
[59] Project, G., . 1000 Genome project datasets.
[60] project OE. Expression project for oncology. 2005. https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE2109. [Accessed 7 September 2020].
[61] Qi GJ, Zhang L, Lin F, Wang X. Learning generalized transformation equivariant representations via autoencoding transformations. IEEE Transactions on Pattern Analysis and Machine Intelligence; 2020.
[62] Ravì D, Wong C, Deligianni F, Berthelot M, Andreu-Perez J, Lo B, Yang G. Deep learning for health informatics. IEEE JBHI 2016;21:4–21. https://doi.org/10.1109/JBHI.2016.2636665.
[63] Reed R, MarksII RJ. Neural smithing: supervised learning in feedforward artificial neural networks. Mit Press; 1999.
[64] Romero A, Carrier PL, Erraqabi A, Sylvain T, Auvolat A, Dejoie E, Legault MA, Dubé MP, Hussin JG, Bengio Y. Diet networks: thin parameters for fat genomics. 2016. arXiv preprint arXiv, 1611.09340.
[65] Rong D, Xie L, Ying Y. Computer vision detection of foreign objects in walnuts using deep learning. Comput Electron Agric 2019;162:1001–10. https://doi.org/10.1016/j.compag.2019.05.019.
[66] Sakurada M, Yairi T. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In: Proceed. Of the MLSDA 2014 2nd workshop MLS data analysis; 2014. p. 4–11. https://doi.org/10.1145/2689746.2689747.
[67] Salimans T, Kingma DP. Weight normalization: a simple reparameterization to accelerate training of deep neural networks. In: NIPS; 2016. p. 901–9. https://arxiv.org/pdf/1602.07868.pdf.
[68] Schmidhuber J. Deep learning in neural networks: an overview. Neur. networ. 2015;61:85–117. https://doi.org/10.1016/j.neunet.2014.09.003.
[69] Sharkawy AN. Principle of neural network and its main types. J. Adv. Appl. Comput. Math. 2020;7:8–19.
[70] Singh V, Baranwal N, Sevakula RK, Verma NK, Cui Y. Layerwise feature selection in stacked sparse auto-encoder for tumor type prediction. In: 2016 IEEE international conference on Bioinformatics and biomedicine (BIBM). IEEE; 2016. p. 1542–8.
[71] Sit MA, Koylu C, Demir I. Identifying disaster-related tweets and their semantic, spatial and temporal context using deep learning, natural language processing and spatial analysis: a case study of hurricane irma. International Journal of Digital Earth 2019. https://doi.org/10.1080/17538947.2018.1563219.
[72] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. JMLR 2014;15:1929–58. https://dl.acm.org/doi/abs/10.5555/2627435.2670313.
[73] Stiglic G, Kokol P. Stability of ranked gene lists in large microarray analysis studies. BioMed Res Int 2010;2010. https://doi.org/10.1155/2010/616358.
[74] Sutradhar R, Barbera L. Comparing an artificial neural network to logistic regression for predicting ed visit risk among patients with cancer: a population-based cohort study. J Pain Symptom Manag 2020;60:1–9.
[75] Suzuki T. Generalization bound of globally optimal non-convex neural network training: transportation map estimation by infinite dimensional Langevin dynamics. 2020. arXiv preprint arXiv, 2007.05824.
[76] Svozil D, Kvasnicka V, Pospichal J. Introduction to multi-layer feed-forward neural networks. Chemometr Intell Lab Syst 1997;39:43–62.
[77] Tian Y, Lu C, Zhang X, Tan KC, Jin Y. Solving large-scale multiobjective optimization problems with sparse optimal solutions via unsupervised neural networks. IEEE transactions on cybernetics 2020.
[78] Tikhonov AN. On the stability of inverse problems. In: Dokl. Akad. Nauk SSSR; 1943. p. 195–8. https://doi.org/10.1007/978-3-642-81472-3_5.
[79] Wager S, Wang S, Liang SP. Dropout training as adaptive regularization. In: NIPS; 2013. p. 351–9. https://arxiv.org/pdf/1307.1493.pdf.
[80] Wang Y, Yao H, Zhao S. Auto-encoder based dimensionality reduction. Neurocomputing 2016;184:232–42. https://doi.org/10.1016/j.neucom.2015.08.104.
[81] Wei C, Kakade S, Ma T. The implicit and explicit regularization effects of dropout. In: International conference on machine learning. PMLR; 2020. p. 10181–92. https://arxiv.org/abs/2002.12915.
[82] Weisberg S. Applied linear regression, ume 528. John Wiley & Sons; 2005.
[83] Wen T, Zhang Z. Deep convolution neural network and autoencoders-based unsupervised feature learning of eeg signals. IEEE Access 2018;6:25399–410.
[84] Werbos PJ. Generalization of backpropagation with application to a recurrent gas market model. N. networ. 1988;1:339–56. https://doi.org/10.1016/0893-6080(88)90007-X.
[85] Werbos PJ. Backpropagation through time: what it does and how to do it. Proc of the IEEE 1990;78:1550–60. https://doi.org/10.1109/5.58337.
[86] Wright RE. Logistic regression. 1995.
[87] Xia Y, Qin T, Chen W, Bian J, Yu N, Liu TY. Dual supervised learning. In: International conference on machine learning. PMLR; 2017. p. 3789–98.
[88] Xie F, Zhang J, Wang J, Reuben A, Xu W, Yi X, Varn FS, Ye Y, Cheng J, Yu M, et al. Multifactorial deep learning reveals pan-cancer genomic tumor clusters with distinct immunogenomic landscape and response to immunotherapy. Clin Canc Res 2020a;26:2908–20. https://doi.org/10.1158/1078-0432.CCR-19-1744.
[89] Xie Y, Wu X, Ward R. Linear convergence of adaptive stochastic gradient descent. In: International conference on artificial intelligence and statistics. PMLR; 2020b. p. 1475–85.
[90] Xin J, Embrechts MJ. Supervised learning with spiking neural networks. In: IJCNN'01. International joint conference on neural networks. Proceedings (cat. Noh01CH37222), IEEE; 2001. p. 1772–7.
[91] Yang J, Ma J. Feed-forward neural network training using sparse representation. Expert Syst Appl 2019;116:255–64.
[92] Yang X, Deng C, Zheng F, Yan J, Liu W. Deep spectral clustering using dual autoencoder network. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition; 2019. p. 4066–75.
[93] Zheng S, Zhao J. A new unsupervised data mining method based on the stacked autoencoder for chemical process fault diagnosis. Comput Chem Eng 2020;135:106755.
[94] Zingaretti LM, Gezan SA, Ferrão LFV, Osorio LF, Monfort A, Muñoz PR, Whitaker VM, Pérez-Enciso M. Exploring deep learning for complex trait genomic prediction in polyploid outcrossing species. Front Plant Sci 2020;11:25. https://doi.org/10.3389/fpls.2020.00025.