ISSN: 2641-6336
DOI: 10.33552/ABBA.2025.06.000637
Annals of
Biostatistics & Biometric Applications
Iris Publishers

Review Article

Deep Learning Signal Processing

Nicolas Nafati[1]*, Francoise Paris[2], and Eric Huyghe[3]

[1]*IRMB-UMR1203, Hospital Saint Eloi Montpellier, France*

[2]*371 Avenue, Du Doyen Gaston Giraud Montpellier, France*

[3]*Département of Urology CHU Toulouse – Rangueil, France*

***Corresponding author:** Nicolas Nafati, IRMB-UMR1203, Hospital Saint Eloi Montpellier, France

Received Date: February 15, 2025

Published Date: February 25, 2025

Abstract

This article focuses on deep learning for random and normalian variables to which we associate toxic and non-toxic characteristics randomly as well. I will give the definition of forward propagation, the definition of back propagation, and also the definition of a two-layer neural network. The results obtained are an image containing toxic and non-toxic samples classified into two groups, I will also give as results the learning curve and the accuracy curve. At the end of this article, I will give the python script that made it possible to obtain these results.

Keywords: Deep learning; normalian variable; toxic; non-toxic; forward propagation; back propagation; neural network; learning curve; accuracy curve

Introduction

Deep learning is a critical subfield of AI (Artificial Intelligence). It is based on machine learning through complex neural networks. This advanced field of machine learning can lead to a deeper understanding of algorithms designed from the neural network of the human brain. Deep Learning makes it possible to achieve complex objectives, ranging from computer vision, object recognition within images, neuro-biological data processing... etc. Deep learning is a rapidly evolving technology as its algorithms interact with complex data by learning to make decisions autonomously and from unstructured data. This allows for no human intervention, unlike machine learning, which requires human intervention to learn the decision- making methodology from large amounts of data. Deep learning is a branch of machine learning that relies on the architecture of artificial neural networks without human intervention, unlike machine learning, which requires human intervention in order to learn the learning methodology.

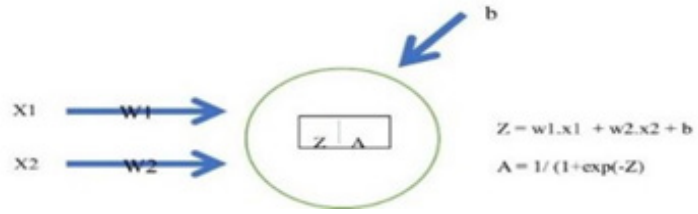
Artificial Neuron

A neuron can be schematized as shown in the following Figure 1.

A neuron is the basic processing unit of a neural network. It is connected to input sources of information (e.g., other neurons) and returns information as output.

Forward Propagation

For a 2-layered neural network, [1,5] the forward propagation is summarized by the calculation of matrices $Z^{[3]}$ and $A^{[3]}$ as shown below Figure 2.



Where:
Z is a linear combination of inputs and weighting points; A being the activation matrix function; x1, and x2 are two vectors columns of input of dimension m; where b is the column vector noise; W1 and W2 is the weighting weights of x1 and x2, respectively, to be optimized.
Figure 1: Artificial Neuron.

$$A^0 = X(input\ vector)$$
$$Z^{[1]} = W^{[1]} * A^{[0]} + b^{[1]}$$
$$Z^{[2]} = W^{[2]} * A^{[1]} + b^{[2]}$$
$$Z^{[3]} = W^{[3]} * A^{[2]} + b^{[3]}$$

With $Z^{[1]} = \begin{bmatrix} W_{11}^{[1]} & W_{12}^{[1]} \\ W_{21}^{[1]} & W_{22}^{[1]} \\ W_{31}^{[1]} & W_{32}^{[1]} \end{bmatrix} * \begin{bmatrix} x_1^{[1]} & x_1^{[2]} & \dots & \dots. & x_1^{[m]} \\ x_2^{[1]} & x_2^{[2]} & \dots & \dots. & x_2^{[m]} \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix}$

And therefore, Z can be written in matrix form as indicated by the following formula:

$$Z^{[1]} = \begin{bmatrix} z_1^{[1](1)} & z_1^{[1](2)} & \dots. & z_1^{[1](m)} \\ z_2^{[1](1)} & z_2^{[1](2)} & \dots. & z_2^{[1](m)} \\ z_3^{[1](1)} & z_3^{[1](2)} & \dots. & z_3^{[1](m)} \end{bmatrix}$$ which belongs to

$$\{IR^{3xm}\}$$

Similarly, we give the activation function in matrix form as follows:

$$A^{[1]} = \frac{1}{1-\exp(-Z^{[1]})} = \begin{bmatrix} z_1^{[1](1)} & z_1^{[1](2)} & \dots. & z_1^{[1](m)} \\ z_2^{[1](1)} & z_2^{[1](2)} & \dots. & z_2^{[1](m)} \\ z_3^{[1](1)} & z_3^{[1](2)} & \dots. & z_3^{[1](m)} \end{bmatrix}$$

which belong to $\{IR^{3xm}\}$

Generalizing, we have the following formulas:

$$A^0 = X(input\ vector)$$
$$Z^{[c]} = W^{[c]} * A^{[c-1]} + b^{[c]}$$
$$A^{[c]} = 1/(1-\exp(Z^{[c]}))$$

Where:

c is the index of the neuron layer

$A^{[c]}$, $W^{[c]}$ and $b^{[c]}$ are the matrix form of the activation, of the weighting functions and the noise vector, all that respectively correspond to the layer c.

The Log Loss Function to Minimize and Back Propagation

The cost function [3,4] to be minimized is given by the following formula:

$$L = \frac{-1}{m} * \sum Y * Log(A+z) + (1-Y) * Log(1-A+z)$$

Where Y is the binary variable known as a vector of dimension m.

Then comes the optimization to minimize the cost function by going through the descent of the gradient. Gradient decent, and updated W and b (noise) parameters are the back propagation such as:

$$w = w - \alpha * \frac{\partial L}{\partial w} \quad \text{and} \quad b = b - \alpha * \frac{\partial L}{\partial b}$$

The optimization process is based on the Gradient decent and on the updated of W and b (noise). This optimization process is such:

$$w^{[1]} = w^{[1]} - \alpha * \frac{\partial L}{\partial w^{[1]}}$$

$$w^{[2]} = w^{[2]} - \alpha * \frac{\partial L}{\partial w^{[2]}}$$

$$b^{[1]} = b^{[1]} - \alpha * \frac{\partial L}{\partial b^{[1]}}$$

$$b^{[2]} = b^{[2]} - \alpha * \frac{\partial L}{\partial b^{[1]}}$$

With $w^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \end{bmatrix} w^{[2]} = \begin{bmatrix} w_{11}^{[2]} \\ w_{21}^{[2]} \\ w_{31}^{[2]} \end{bmatrix}$

$$b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix} \quad b^{[2]} = \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \\ b_3^{[2]} \end{bmatrix}$$

For a c-layered neural network, the forward propagation is summarized by the calculation of matrices $Z^{[c]}$ and $A^{[c]}$ as shown below:

$$A^0 = X \ (input\ vecteur)$$

$$Z^{[c]} = W^c * A^{c-1} + b^c$$

$$A^{[c]} = 1/(1-\exp(Z^{[c]}))$$

Where:

c is the index of the neuron layer.

$A^{[c]}$, W^c are respectively the matrices of activation and of weighting functions at layer c

b^c is the noise vector corresponding to layer c

Next comes the partial derivatives of the Log Loss cost with respect to w and b for two layers:

$$L = \frac{-1}{m} * \sum Y * Log(A^{[2]} + z) + (1-Y) * Log(1 - A^{[2]} + z)$$

$$\frac{\partial L}{\partial w^{[1]}} = \frac{\partial L}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial w^{[1]}}$$

$$\frac{\partial L}{\partial w^{[2]}} = \frac{\partial L}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial w^{[2]}}$$

$$\frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial b^{[2]}}$$

$$\frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial b^{[1]}}$$

For a c-layer neural network, we have:

$$dz^{[c]} = A^{[c]} - y$$

$$dw^{[c]} = \frac{1}{m} \sum dz^{[c]} * A^{[c-1]^T}$$

$$db^{[c]} = \frac{1}{m} \sum dz^{[c]}$$

$$dz^{[c-1]} = w^{[c]^T} * dz^{[c]} * A^{[c-1]} * (1 - A^{[c-1]})$$

Algorithm of Deep Learning

```
_ def neuronal_network(X, y, hidden_layers=(32, 32, 32), learning_rate=0.1, n_iter=1000):
    np.random.seed(0)
    ######## Initialisation
    dimensions = list(hidden_layers)
    dimensions.insert(0, X.shape[0])
    dimensions.append(y.shape[0])
    print('dimensions after insert and append', dimensions)
    parametres = initialisation_param(dimensions, X, y)

    train_loss = []
    train_acc = []

    for i in tqdm(range(n_iter)):
        activations = forward_propagation(X, parametres)
        gradients = back_propagation(y, activations, parametres)
        parametres = update(gradients, parametres, learning_rate)
```

```
    train_loss = []
    train_acc = []

    for i in tqdm(range(n_iter)):
        activations = forward_propagation(X, parametres)
        gradients = back_propagation(y, activations, parametres)
        parametres = update(gradients, parametres, learning_rate)


    if i % 10 == 0:
        C = len(parametres) // 2
        train_loss.append(log_loss(y, activations['A' + str(C)]))
        y_pred = predict(X, parametres)
        y_pred = y_pred[3, :]
        y_pred = y_pred.T
        current_accuracy = accuracy_score(y.flatten(), y_pred.flatten().T)
        train_acc.append(current_accuracy)
# results visualisation
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(10, 4))
ax[0].plot(train_loss, label='train_loss')
ax[0].legend()

ax[1].plot(train_acc, label='train_acc')
ax[1].legend()
# visualisation(X, y, parametres, ax)
plt.show()
return parametres
```

Learning Curve

The obtaining learning curve (Figure 3) is downward at the beginning and flat towards the end for these data, with the cost per unit represented on the Y axis and iterations index on the X axis. As learning increases, the cost per unit of output initially decreases before flattening as it becomes more difficult to increase the efficiency gained through learning process.

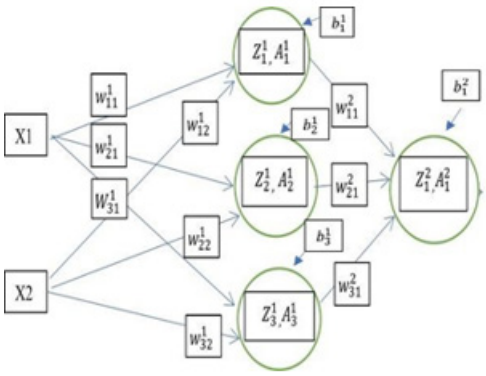


Figure 2: Two-layers neural network forward propagation.

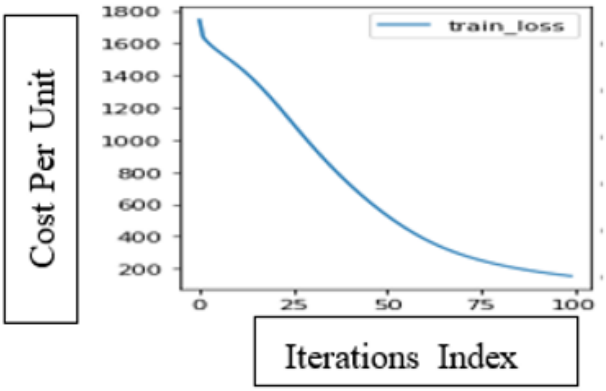


Figure 3: Learning curve.

Validation Curve

Exactitude measured by the deviation of the results from the "true" value. Combination of fidelity and exactitude Y is the total error. It is not because we have a faithful and fair method that it is automatically exact [2,6]. It is therefore necessary to check the exactitude at the end Figure 4. Exactitude is a metric for evaluating the performance of classification models with 2 or more layers. Exactitude can be translated as "precision" in French. Like other metrics, exactitude is based on the confusion matrix. As a reminder, the confusion matrix is composed of 4 values (True Negative, False Negative, False Positive, True Positive).

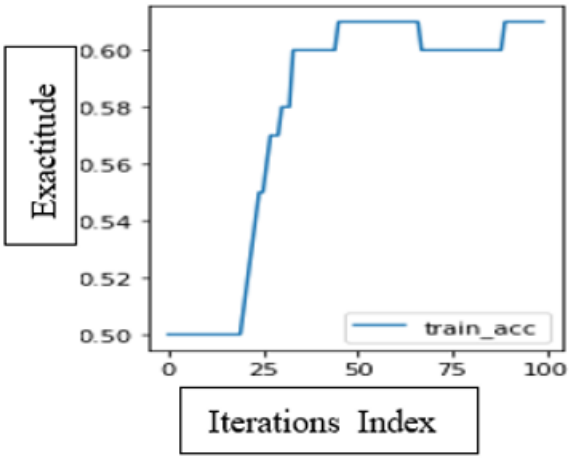


Figure 4: Exactitude curve.

Deep Learning Image Result

Accuracy makes it possible to describe the performance of the model on positive and negative individuals in a symmetrical way. It measures the rate of correct predictions on all individuals Figure 5.

Conclusion

Machine learning and deep learning are two types of artificial intelligence (AI). Machine learning is AI that can adapt automatically with minimal human interference, and deep learning is a subset of machine learning that uses neural networks to mimic the learning process of the human brain. Our algorithm uses Deep Learning to classify input data into two groups, toxic and non-toxic samples, as shown in Figure 5.

Deep Learning vs Machine Learning: What Are the Differences?

Machine learning and deep learning are two types of artificial intelligence. Machine learning is AI that can adapt automatically with minimal human interference, and deep learning is a subset of machine learning that uses neural networks to mimic the learning process of the human brain. There are several major differences between these two concepts. Deep learning requires larger

volumes of training data, but learns from its own environment and mistakes. On the contrary, machine learning allows training on smaller datasets, but requires more human intervention to learn and correct mistakes. In the case of Machine Learning, a human must intervene to label the data and indicate its characteristics. A deep learning system, on the other hand, tries to learn these characteristics without involvement.

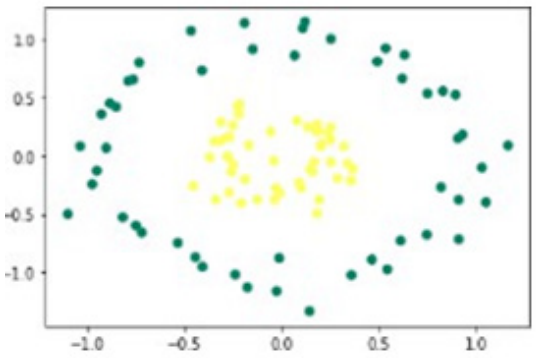


Figure 5: Image result composed of toxic (green) and not toxic samples (yellow).

Script With Python Language

```
def initialisation(dimensions, X, y):
    input('dans initialisation_param')
    parametres = {}
    C = len(dimensions)
    X = np.array(X)
    y = np.array(y)
    for c in range(1, C):
        print('dimension C', c)
        x_NbLig = X.shape[0]


        y_NbCol = X.shape[1]
        parametres['W' + str(c)] = X[c: x_NbLig][ c: y_NbCol]
        parametres['b' + str(c)] = y[c: y_NbCol]
    return parametres

def  forwardpropagation(X, parametres):
    activations = {'A0': X}
    C = len(parametres) // 2
    print('C', C)
    for c in range(1, C+1):
        print('index c', c)
        Activation = []
        Activation = np.array(activations['A' + str(c-1)])
        Activation = Activation.T
        Matrix_W = np.array(parametres['W' + str(c)])
        print('Activation shape after transposition', Activation.shape)
        print('W matrix', X.shape)
        input('Type of activation var ')
        Matrix_W = Matrix_W.astype(float)
```

```
                Activation = Activation.astype(float)
                print('size Matrix W', Matrix_W.shape)
                print('Activation.shape', Activation.shape)
                input('debug just before the product')
                Vect_B = parametres['b' + str(c)]
                 Z = np.dot(Matrix_W, Activation)
                 Z = Z + np.array(Vect_B, dtype=float)
                 Z.shape', Z.shape)
                 print('Z matrix issue du produit matrcielle', Z)
                 activations['A' + str(c)] = np.array(1/(1+np.exp(-Z))

            return activations


activations = forwardpropagation(X, parametres)
  for key, val in activations.items():
     print(key, val.shape)

def backpropagation(y, activations, parametres):
   print(y.shape)
   m = y.shape[1]
   C = len(parametres) // 2
   dZ = activations['A' + str(C)] - y
   gradients = {}
   for c in reversed(range(1, C+1)):
      gradients['dW' + str(c)] = 1/m * np.dot(dZ, activations['A' + str(c-1)].T)
      gradients['db' + str(c)] = 1/m * np.sum(dZ, axis=1, keepdims=True)
      if c > 1:
         dZ = np.dot(parametres['W' + str(c)].T, dZ) * \ activations['A' + str(c-1)] * (1-activations['A' +

                                                    str(c- 1)])
                                                       return gradients


gradients = backpropagation(y, activations, parametres)
for key, val in gradients.items():
   print(key, val)

def log_loss(A, y):
   z = pow(10, -15)
   return 1 / len(y) * np.sum(-y * np.log(A + z) - (1-y) * np.log(1-A + z))

def update(gradients, parametres, learning_rate=0.1):
   C = len(parametres) // 2
   for c in range(1, C+1):
      parametres['W' + str(c)] = parametres['W' + str(c)] - \ learning_rate * gradients['dW' + str(c)]
      parametres['b' + str(c)] = parametres['b' + str(c)] - \ learning_rate * gradients['db' + str(c)]
   return parametres


                def predict(X, parametres):
                   activations = forward_propagation(X, parametres)
                   A2 = activations['A2']
                   A2 = (A2 >= 0.5)
                   print('A2 in predict', A2)
                   print('A2 shape in predict', A2.shape)
                   return A2
```

```
A2 = predict(X, parametres)

def neuronal_network__TwoLayers(X_train, y_train, n1, learning_rate=0.1, n_iter=1000):
    # Initialisation
    n0 = X_train.shape[0]
    n2 = y_train.shape[0]
    parametres = initialisation(n0, n1, n2)
    train_loss = []
    train_acc = []
    for i in range(n_iter):
        activations = forward_propagation(X_train, parametres)
        gradients = backpropagation(X_train, y_train, activations, parametres)
        parametres = update(gradients, parametres, learning_rate)

        if i % 10:
            train_loss.append(log_loss(y_train, activations['A2']))
            y_pred = predict(X_train, parametres)
            current_accuracy = accuracy_score( y_train.flatten(), y_pred.flatten())
            train_acc.append(current_accuracy)



                    plt.figure(figsize=(14, 4))
                    plt.subplot(1, 2, 1)
                    plt.plot(train_loss, label='train_loss')
                    plt.legend()

                    plt.subplot(1, 2, 2)
                    plt.plot(train_acc, label='train_acc')



                    plt.legend()
                    plt.show()

                    return parametres
```

References

1. Amita Kapoor, Antonio Gulli, Sjit Pal, François Colled (2022) Deep Learning with Tensor Flow and Keras. Third Edition: Build and deploy supervised, unsupervised deep, and reinforcement learning models.
2. Catherine Y (2015) Exactitude et intervalles statistiques en validation de méthode. 17th International Congress of Metrology.
3. Collobert R (2011) Deep learning for efficient discriminative parsing. Artificial Intelligence and Statistics 15: 224-232.
4. Amitha M, Arul A, Sivakumari S (2021) Deep Learning Techniques: An Overview. Advanced Machine Learning Technologies and Applications PP. 599-608.
5. Schmidhuber J (2015) Deep learning in neural networks: An overview. Neural Networks 61(3): 85-117.
6. Gulshan V, Peng L, Coram M, Stumpe MC, Wu D, et al. (2016) Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. JAMA 316(22): 2402-2410.