University of Kasdi Merbah Ouargla
Faculty of New Technologies of Information and Communication
Department of Computer Science

Assignment on

Artificial Neural Networks

Course title

Data analysis

Submitted by

Salah Belila
&
Youcef Gasmi

To

Prof. M.L. Kherfi
Course lecturer

At

February 4th, 2021

Abstract

Artificial neural networks are known to be powerful learning algorithms. This report explains ANNs and their architecture along with implementing an ANN from scratch and applying it on the handwritten digits classification problem.

Keywords: Deep learning, ANN, Backpropagation, Classification, Multi-label classification.

I. Introduction

An artificial neural network or ANN for short is a machine learning algorithm that has a wide range of applications. The ANN can be both a supervised or an unsupervised learning algorithm. However, in both cases, the essential idea does not differ from one another. This report goes through the details of how a neural network works, how it is trained and gives a mathematical definition for the used learning algorithm. Finally, we apply a simple neural network on a real problem and discuss the results. Of course, this implies that we will be implementing the ANN ourselves, we will be using python and NumPy for that. Although the ANN used for this little experiment is rather simple, one would notice later in this report that our implementation can handle an arbitrary number of user-defined layers.

II. ANN and its usage

a neural network is defined as A computing system made up of several simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs[1].

Generally, an artificial neural network consists of several layers. These layers consist of several nodes, each layer is connected to its preceding layer with weights. Neural networks can be used as a supervised learning algorithm such as in the example explained later in this report or an unsupervised learning algorithm such as the auto-encoder.

III. Working mechanism and training

The input is a group of nodes, these nodes affect the values of the hidden layer nodes, which in turn affect the values of the output layer through a set of weights between each of the two consecutive layers. We get the nodes of the next layer by calculating the sum of the product of the previous layer nodes multiplied by the weights between these two layers. This process is repeated until we reach the last layer and the final output is produced. Then, given the network output and the corresponding ground-truth, the cost is calculated. Thereafter, given the cost, the network will start updating the weights from the last to the first layer. Thus, it can be said that training an artificial neural network takes place through two stages or two algorithms.

The first phase in which the output and the loss are calculated is called forward propagation or forward pass. And the second phase is responsible for improving the weights to obtain an output as close as possible to the ground truth or the target.

1. The forward pass

The network is given an input (one of the dataset samples) and its weights and biases are randomly initialized. Each layer produces an output to be passed as input to the next layer, this output is given by

$$a_j^l = f\left(\sum_{k=0}^{n} w_{jk}^l \, a_k^{l-1} + b_j^l\right)$$

where w_{jk}^{l} is the weight, a_k^{l-1} is the output from the previous layer, l is the layer index and f is the activation function.

Now comes the role of the cost function. The cost function calculates how bad our network is performing. So the closer to the minima of the cost, the closer the network is to the optimal performance.

2. Backpropagation

Backpropagation is short for backward propagation of error. The backpropagation algorithm uses the gradient descent[1] to update the weights iteratively throughout the training, hoping that the network will converge to the minima[2]. Backpropagation, at each training iteration, updates the weights of layer l as follows

$$w_t^l = w_{t-1}^l - \frac{\partial c_{t-1}}{\partial w_{t-1}^l} \cdot \alpha$$

Where the term $\frac{\partial C}{\partial w}$ is the cost derivative w.r.t the weight matrix w, α is the learning rate, it controls how quickly the model is adapted to the problem and t denotes the training iteration.

The derivative of the cost function w.r.t the weights is given in any layer as follows

$$\frac{\partial c}{\partial w^l} = \frac{\partial c}{\partial z^l} \cdot \frac{\partial z^l}{\partial w^l} \tag{1}$$

Where W^l is the matrix of weights between the layer l and the next layer $l+1$, z^l is defined by

$$z^l = w^l \cdot a^{l-1} + b^l$$

Let

$$\delta^l = \frac{\partial c}{\partial z^l} \tag{2}$$

δ^l is called the error of layer l.

from **(1)** and **(2)** we have

$$\frac{\partial c}{\partial w^l} = \delta^l \cdot \frac{\partial z^l}{\partial w^l} \tag{3}$$

For the output layer ($l = L$)

$$\delta^L = \frac{\partial c}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \tag{4}$$

Where a^L is the activation function of the output layer L and z is a parameter of a^L.

For any other layer where $l < L$

$$\delta^l = ((w^{l+1})^T . \delta^{l+1}) \odot a'(z^l) \tag{5}$$

Proof:

From equation **(2)** we have

$$\delta_j^l = \frac{\partial c}{\partial z_j^l}$$

[1] We are not going through the gradient descent algorithm due to the number of pages' limit. However, a quick google search will do the job.

[2] The network does not need to converge to the minima. In practice, even the convergence to some good enough local minimum is considered a good result.

$$= \sum_k \frac{\partial c}{\partial z_j^{l+1}} \cdot \frac{\partial z_j^{l+1}}{\partial z_j^l}$$
$$= \sum_k \delta_k^{l+1} \cdot \frac{\partial z_j^{l+1}}{\partial z_j^l}$$

We also have

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1}$$

Which implies

$$\frac{\partial z_j^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} a'(z_j^l) = (w_{jk}^{l+1})^T a'(z_j^l) \quad (6)$$

we substitute **(6)** in **(5)** to obtain

$$\sum_k (w_{jk}^{l+1})^T \delta_k^{l+1} a'(z_j^l) \quad (7)$$

As for the biases, it is a simple derivative

$$\frac{\partial c}{\partial b^l} = \delta^l \cdot \frac{\partial z^l}{\partial b^l} = \delta^l \quad (8)$$

The backpropagation algorithm can be summarized in these five equations:

$$\frac{\partial c}{\partial w^l} = \delta^l \cdot \frac{\partial z^l}{\partial w^l} \quad \text{BP.(1)}$$

$$\delta^L = \frac{\partial c}{\partial a^L} \cdot \frac{\partial a^L}{\partial z^L} \quad \text{BP.(2)}$$

$$\delta^l = ((w^{l+1})^T . \delta^{l+1}) \odot a'(z^l) \; \forall \; l < L \quad \text{BP.(3)}$$

$$\frac{\partial c}{\partial b^l} = \delta^l \quad \text{BP.(4)}$$

$$w_t^l = w_{t-1}^l - \frac{\partial c_{t-1}}{\partial w_{t-1}^l} \cdot \alpha \quad \text{BP .(5)}$$

IV. Using ANN for handwritten digit classification

Now, that we know how neural networks work, we will try applying them on a real problem to see how effective they actually are. We will be using an ANN to predict the digit given a handwritten digit image as input. First, we will explain the used dataset. Then, we will implement the ANN. Our ANN implementation should accept an arbitrary number of user-defined layers.

1. The MNIST dataset

The dataset that will be used is known as the MNIST dataset. This dataset is comprised of labeled images. Each image is of a handwritten digit, whereas the label is an integer value of the corresponding digit. So, at each training iteration, the network will take an image as its input and the corresponding label as the target. The MNIST

dataset contains 60K training samples and 10K test samples.

2. ANN implementation

The implementation of ANN simply means implementing the forward pass algorithm and the backpropagation algorithm. Then, the ANN class must have: 1) a forward method, 2) a backward method. The forward method will do the forward pass then will calculate the error. While the backward method will do the backward pass based on the calculated error. However, in this approach, we will have to hardcode the error backpropagation given by eq. (4), and this does not seem like a general solution because we aim to implement an ANN that takes an arbitrary number of layers.

As an alternative, we will instead, implement the Layer class that have its own forward and backward methods. Then, the ANN class will have a list of layers each of which has its own forward and backward methods. And by iteratively calling the forward method in each of these layers, we can do a forward pass, similarly for the backward pass. Our workflow will be as follows:

The Layer class:

- The constructor will take input and output dimensions of the layer as arguments (this will also be used to define the weight matrix). Then randomly initialize the weight matrix.
- A forward method that takes input and returns its output to be passed to the next layer.
- A backward method that takes the error from the next layer (δ^{l+1}), then, updates the weights and returns the error of the corresponding layer (δ^{l}).

The ANN class:

- The constructor will take a list of layer instances.
- A forward method that iteratively calls the forward method in the layers list to do a full forward pass.
- A backward method that iteratively calls the backward method in the layers list to do a full backward pass.

Finally, we add an optional class called Function. This class will let us implement any mathematical function f, thus this class will have two methods:

- The activate method, takes X as input and returns $f(X)$.
- The differentiate method, returns $f'(X)$.

The source code can be found in GitHub[3].

3. Results

After running several experiments by changing the network's hyperparameters, namely, the learning rate and the hidden dimension, we found that the combination of learning rate = **0.07** and hidden dimension = **400** gives the best accuracy of **97.8%**. We ranged the learning rate between 0.01 and 0.1 by a step of 0.01. And for each of these values, we trained the network for 6 epochs for every hidden dimension of the following set: {50, 80, 120, 150, 200, 300, 400, 500}. We used the sigmoid function for both

[3] https://github.com/SalahBelila/ann

layers as an activation function. While the mean square error function was used as a measure of cost evaluation.

Of course, one could go and try changing even more hyperparameters such as the number of the hidden layers and the activation function for each layer. However, this seems a bit of overkill for a simple task such as the one we had.

V. Conclusion

In this mini-project, we discussed the working mechanism of the ANN and its implementation from scratch. However, one can notice that we applied the ANN only on a classification problem. Since no matter what the problem is, both the forward pass and the backpropagation algorithms will remain applicable. Another thing covered in this report was how the activation/loss functions were chosen. There are no clear rules for that, but an activation function must have some characteristics:

1. An activation function must squash the values between some interval to prevent the gradient from vanishing/exploding.
2. An activation function must be suitable to define the notion of weight and correlation between neurons.
3. And of course an activation function must be continuous.

However, an activation function can be chosen on another basis aside from the characteristics mentioned above. E.g. for the above classification problem we used the sigmoid in the last layer, however, the softmax activation function should have been a better choice for a classification problem such as that[4]. Whereas the choice of the loss function mainly depends on the type of the problem (whether regression or classification). However, as a rule of thumb, a good loss function must not be fluctuant, to minimize the possibility of the loss being stuck in a local minimum.

Thus far in this section, we did not discuss the implementation. And there is not much to discuss, but, our implementation has one crucial problem. That is, it does support mini-batch training. Thus the first and most important improvement that should be done in the implementation is adding the mini-batch training functionality.

[4] For a better understanding of why the softmax was a better solution we recommend visiting: https://en.wikipedia.org/wiki/Softmax_function

References

1. Huang LF. *Artificial Intelligence*. Vol 4.; 2010. doi:10.1109/ICCAE.2010.5451578

2. Nielsen M. *Neural Networks and Deep Learning*; 2015. http://neuralnetworksanddeeplearning.com/