Deep Learning - A brief study of Main Approaches and Architectures

Bryan Mauricio Reinoso Cevallos

[1]Student from Burgos's Polytechnic School

brc0007@alu.ubu.es; cevallos.reinoso10@e-uvt.ro

Abstract. *The main goal of this paper is to do a brief study and comparison of main architectures of Deep Learning.*
Firstly, a state of the art chapter is goin to be presented where I am going to do a brief introduction to the Deep Learning. Because this paper is about a very dificult subject, a deep state of the art is almost impossible to do in 6 to 12 pages, so in this first chapter is going to be a brief introduction of the main topic and the differents main approaches of this matter.
Secondly, a more detailed chapter with the main architectures is going to be presented. Also, some problems and models are going to be written in this chapter.
Thirdly, I am going to write a deep report about the application made by my classmate and me and the result we obtained. This application is going to be explained in detail. Also, a report of the results is going to be presented.
Lastly, a chapter with some conclusions and future work in this topic.
Keywords*: Deep Learning.*

1. Introduction: State of the art

Firstly, in order to understand the main purpose of this topic, we also have to understand why Deep Learning is called that and what kind of problems are going to be addressed in this field.

So, We have to understand that when we talk of Deep Learning, we are talking of a field which address very complex problems. In order to solve this kind of problems a knowledge representation is needed. This representation is based in the real world knowledge and its purpose is to be understandable by the computer. Then, we are going to be able to build some architectures and algorithms to address these problems under this representation.

Therefore, this representation will allow us to model the problem and the training examples. The training examples are a representation of the problem in a particular case and, if we are talking of supervised training, a label with the solution of the problem for that particular case.With this training examples under the representation, we have the codified knowledge to build the algorith and the architecture which is going to learn from that information.

Because this information represents the knowledge of a specific task, we can say that we need a deep model that can capture the information of the problem under its representation and build a complex function with which to generalize the knowledge provided by the training examples.

Thus, in order to address these problems to achieve acceptable results, researchers began to model better and deeper algorithms and architectures of which major ones are going to be presented.

1.1. Neural Networks

One of the most popular models for Deep and Machine Learning. Because this is the method used for the application its presentation is going to be described in one section below (See Section 2).

1.2. Boltzmann Machines

1.3. Other honorable mentions

After the presentation of Boltzmann Machines and Neural Networks, I think is interesting to present another three algorithms that are also powerful and could be used to solve complex problems.

1.3.1. Linear Regression

Linear Regression is one of the mos used algorithms to predict a certain value or to classify from a serie of features that represent every input example. The intuition of this process is to draw a line through the examples which minimize the cost function. In order to understand this intuintion we can see the Figure 1.

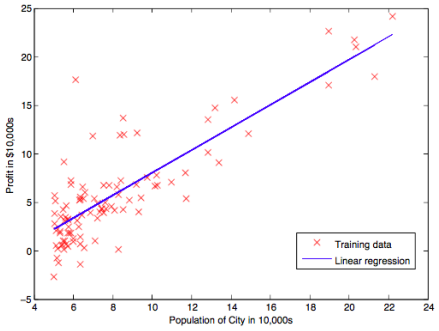

Figure 1. Linear Regression

So, in this algorithm we have two principal components. One is the cost function whose minimization is the objetive of Linear Regression. The second one is the hypotesis which is a linear function that represents the line to be drawn whose components are the θ value.

Firstly, the cost functions is defined by the sum of the quadratic difference between the hypotesis and the objetive value of all training examples:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{1}$$

Secondly, the hypotesis that is only a equation for a line but with the θ value that is going to change in the training in order to get the best fit for the data.

$$h_\theta(x) = \theta^T x = \theta_0 + \theta_1 x_1 \tag{2}$$

And this is Linear Regression algorithm which consist in draw the straight line through the training examples that minimize the cost function. Now, to obtain the best value θ for the problem, we have to use another algorithm and the most popular is Gradient Descent. And, in particular, we are going to present Batch Gradien Descent (see Section 1.3.3).

1.3.2. Logistic Regression

Logistic Regression has almost the same intuition as Linear Regression but in this case we are not trying to draw a straight line but a curve (See Figure 2). Also, Logistic Regression

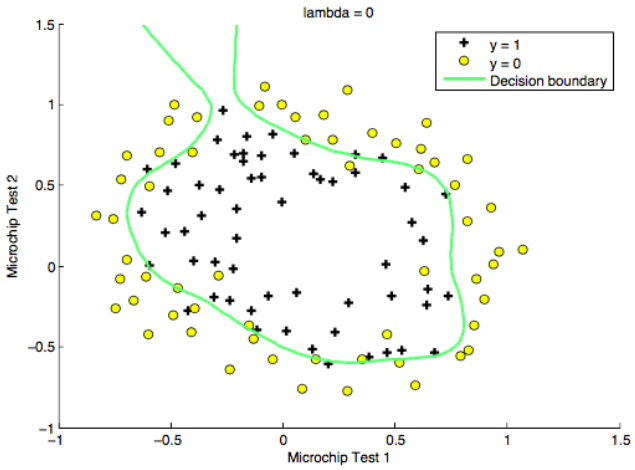

Figure 2. Logistic Regression

algorith has two components in its process to obtain the best curve to fit the data.

The first one is the cost function. The formula of the cost function for Logistic Regression seems harder than the formula for Linear Regression, but it is only the logarithmic function of the hypotesis or one minus the hypotesis dependind if the expected value is one or cero. So if the expected value is cero, then the cost function for that example is the logarithmic funtcion of one minus the hypotesis for that example, and, if the expected value is one, the cost functions is the logarithmic function of the hypotesis. The total cost function is the sum of the cost functions of all training examples. The formula is:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))] \tag{3}$$

The intuition for the formula is that, when you are trying to predict one, if the hypotesis is close to one the algorithmic function will return a number close to cero. And, if you are trying to predict cero, when the hyposetis is close to cero, we subtract a lower amount from the one value and we are requesting the logarithmic function value for a number

close to one that is cero. So, when we are far from the expected value, the cost funciont is going to be higher.

Now, the second component, is the hypotesis. Unlike in Linear Regression, in this algorithm we are trying to draw a curve that fits the data, so, the hypotesis is going to be a nonlinear function. In this case we are talking about the sigmoid function.

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} \tag{4}$$

And, like in Linear Regression, to calculate the θ value we have to perform an update for the θ using another algorith. This algorith can also be the Batch Gradient Descent (see Section 1.3.3).

1.3.3. Batch Gradient Descent

Batch Gradient Descent is an algorith that determines the way we perform an update of the theta values. In every iteration of the training an uptdate is going to be performed. In Batch Gradien Descent we use all training examples to calculate the update needed to be performed.

So, in each iteration of the algorith, batch gradient descent is going to perform the update:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{5}$$

We have to know that this update is done simultaneously in θ_j for all j. With this update our θ parameter come closer to the optimal and, in result, our cost function $J(\theta)$ decrease achieving the lowest cost.

In order to increase the performance of the algorithm, it is common to use the gradient values. these gradients are calculated at the same time as the cost function increasing the overall performance.

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{6}$$

It is noteworthy that there are some other updating algorithms to increase more the performance.

2. Model used in the application: Neural Networks

It is well known that when we speak of Machine Learning and Deep Learning, one of the most widespread models are Neural Networks. Neural Networks, as you probably know, are a model based on the biological structure of brain neurons and their interconnection.

With this model, we are able to build some neural networks capable of learning complex nonlinear functions. These functions are an abstraction of the knowledge extracted from the training examples. With this complex nonlinear function we should be able to generalize knowledge to other new examples never seen by the neural network.

Neural Networks are composed of artificial neurons, which have an activation function, a series of inputs and an output. The process that this neuron does is to receive these inputs, and each input has a weight that will multiply with it. After this, a summation is made between all the previous results. This summation is given to the activation function that will return the output of the neuron. And, because the Neural Network has several neurons, it can abstract the complex nonlinear funcions which is composed by all calculations that every neuron do. An illustrated example of this operation can be seen in the Figure 3.

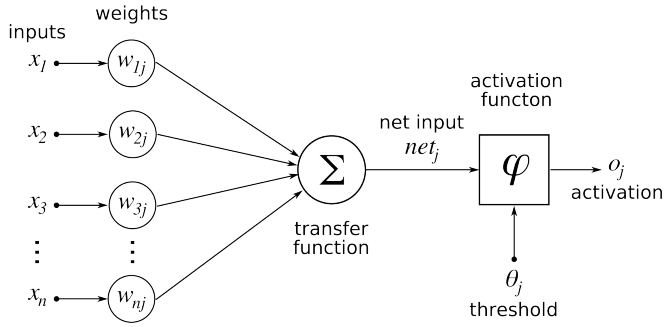

Figure 3. Artificial Neuron

Knowing how artificial neurons works, we have to know how this neurons are organized inside the neural network. The neural network architecture is divided in, at least, two layers. One layer is called input layer, and the other one is called ouput layer. And, between this two layers, we can add one or more hidden layers. Of course we have to hae in mind that the bigger the numer of layers is, the more calculations and training complexity we have. We can se an example of a neural network with multiple layers in Figure 4. Also, we have to think of the possibilitie of concatenate neural networks in order to solve more difficult problems or to build a better pipeline for the problem.

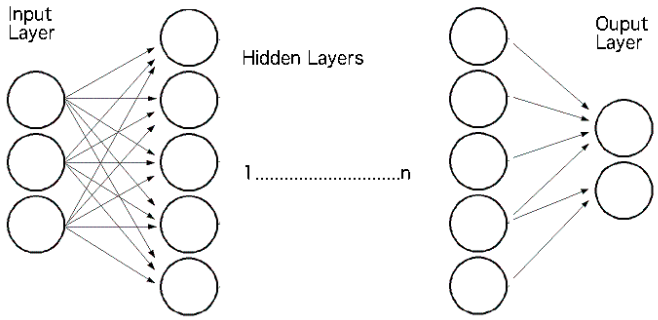

Figure 4. Artificial Neuron

Now, the matter is how a Neural Network can abstract the knowledge from the problems. The answer lies in the representation of the problem and, more specifically,

in how we tune this model. So, firstly, we, as Deep Learning Engineers, must build a good representation of the problem for being understood by the computer. And lastly, we shloud tune the Neural Network in order to obtain the best results. This can be done by changing the number of layers, the way the layers are interconnected, the activation functions of the neurons, the restrictions of the supervised training, etc... Of course this procces of tunning is not easy, we can't be sure which configuration is going to work best and there is not a way to predict this.

Neural Networks have a lot of applications which the main ones are Character and Image recongnition, Image Compression, Stock Market Prediction and Some medicine applications such as cancer prediction.

2.0.1. Multilayer

3. Description of the application and its resuslts

There are three main components to be described in this section. The first one is the architecture and tool used and developed for this application. The second one is the data used to train the model and to test it. And, the last one, is the results obtained while and afteher the training of the Neural Network.

3.1. Architecture and tool

Firstly, we choose the Theano tool because is one of the most popular tools and is for Python which is a very malleable and easy used. Also, in order to manage the arrays in a more convenient way we use Numpy for python.

3.1.1. Theano[1]

This tool for Python is one of the most populer ones to develop Deep Learning models. This is becaus Theano allows the user to run his code in the GPU and, also, it does a lot of compilations in C in order to achieve a better performance.

Theano also has a lot of functions and funtionalities that makes easy the job of implementing Deep Learning models. Theano allow us to work with expressions and representations of values and functions. So, if we want to work with Theano we have, at least, to know that this library works in other way. We have to define variables with types of Theano and used that variables to build functions. Also we have to know how shared variables works, because we used shared values for the weights.

The variables we used in our application are implementing as follows:

```
1    x = T.dvector('x')
2    y = T.dvector('y')
3    theta1 = theano.shared(np.array(np.random.rand(14,10), dtype=
         theano.config.floatX))
4    theta2 = theano.shared(np.array(np.random.rand(11,5), dtype=
         theano.config.floatX))
5    theta3 = theano.shared(np.array(np.random.rand(6,2), dtype=
         theano.config.floatX))
```

But we have to know that these variables are like the python's variables. These variables are a representation of what type of value is expected by Theano when we use the variables in the expression to define formulas, the cost function formula for example.

```
cost_value=T.sum(-y*T.log(layer3)-(1-y)*T.log(1-layer3))
```

This expression can be converted into a funtionc to be used in the future.

```
cost_function_test=theano.function(inputs=[x,y],outputs=[
    cost_value])
```

After implementing the function we can use it to get the cost function. We have to define the inputs using Theano variables. And the ouput can be a normal Python function or a Theano expression. Also, we can determine some updates to be done when the function is called.

```
cost_function=theano.function(inputs=[x,y],outputs=[
    cost_value],updates=[(theta1,gradient(cost_value,theta1,
    alpha)),(theta2,gradient(cost_value,theta2,alpha)),(
    theta3,gradient(cost_value,theta3,alpha))])
```

If the update is done on variables that are not shared, then it has to be in the inputs. The ouputs are pairs with the variable to be updated and the value to update, can also be functions to calculate the new value. Now, we can use this function as a step of the train because we process one example and update the weights with the information of that example.

That was the basic information we used to implement our application with the Theano tool. For more information refer to the Theano Documentation.

3.1.2. Architecture

In order to manage this problem we decided to use the well known model of Neural Network.

The Neural Network of our example has 3 layers. The number of layers was decided for convention between the authors of the application and after testing the application with different numbers of layers and neurons. The best and most interesting results were obtained with the three layers we mentioned and with 10, 5 and 2 neurons respectively. The last layer, ouput layer, has 2 neurons, 1 neuron for each class.

The activation function for the neurons is the sigmoid function. This function was decided because for classification is one of the most populars ones. This is because the sigmoid function returns a value between 0 and 1, the values we want because we are predicting probabilities. So, in the ouput layer each layer is going to predict the probability of one example to be the class that the neuron represents.

For the training we decided to use the stochastic gradient descent algorithm. This algorith is described in a previous section so to know more about it see the Section PENDING.

In the thery, for stochatic gradient descent we have the cost funtion described in its section, but in this case and using the theory of Neural Networks for classification we

used the cost function described in the section of Neural Networks.

And, the last detail to be mentioned is that we have a random values for the θ of the layers at the start of the application. This is noteworthy because one of the greatest advances in Deep Learning is the unsupervised pre-training[4, 3]. We decided this random initialization because we consider that the problem is not hard enough and the training examples are not enough to need this.

3.2. Dataset Used: Heart Disease Data Set [5]

For this application we used a dataset with the data of the heart of some pacients and the information if they had a heart disease or not.This data was taken from four hospitals and it contains a total 76 features for each pacient, but the description said that only 14 features are used in the other works. Also, one of the dataset with 76 features is corrupted so we took the processed datasets with 14 features. [7, 8, 2, 6]

The application take the data from the datasets and storage it to be treated. We do a shuffle of all training examples to mix the pacients of all hospitals. After that we prepare the data to be processed by the neural network.

Firstly, we split the data into two arrays. The first one is filled with the input examples with 13 features, and the second one is filled with the ouputs or the expected value with 1 feature.

```
1 shuffle(data) #Shuffle the data of all datasets
2 for i in data:
3       inputs.append(i[:len(i)-1]) #Split the inputs
4       outputs.append(i[len(i)-1]) #Split the expected outputs
```

The data in the ouputs array should be only 0 or 1 but some examples have the values 2 and 3 also, so we took all value greater than 0 as 1.

```
1 for i in range(len(x)): #For values greater than 1 in the
     predicted value, we asume them as 1
2         if x[i][len(x[i])-1] > 1:
3             x[i][len(x[i])-1]=1
```

Now, the process is to convert the ouputs into an array because we are gonna have 2 neurons to predict two classes. The class heart disease ($[1, 0]$) and the class no heart disease ($[0, 1]$).

```
1     for i in outputs:                    #Converting the outputs
         to be predicted by 2 neurons
2         if i==1:
3             outputsF.append([1,0])
4         else:
5             outputsF.append([0,1])
```

Secondly, is to apply a mean normalization over tha features of the inputs examples. This process is done in order to have a better update of the θ and to reduce the difference between the features. Some features have a high values and other have only values of 0 and 1. The mean normalization is done with the next formula:

$$\forall x \in X, \forall f \in, x^{(f)} = \frac{x^{(f)} - min(X^{(f)})}{max(X^{(f)}) - min(X^{(f)})} \tag{7}$$

We have that, for each input x in the set of inputs X, we do the process of normalization for each feature f in the x example. The normalization is done with the difference between the f value in the example x and the minimum value over all values of the feature f in the set of inputs X divided by the difference between the maximum value over all values of the feature f in the set of inputs X and the minumum described.

And lastly, the process is to divided the full dataset into two subdatasets. The first dataset is for the training process and it has the 70% of all examples. The second dataset is for testing and it has the 30% of all examples. With that process we have the data ready to be used. To end this section a list of the features used and its descriptions is going to be presented.

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
    (a) Value 1: typical angina
    (b) Value 2: atypical angina
    (c) Value 3: non-anginal pain
    (d) Value 4: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar ¿ 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
    (a) Value 0: normal
    (b) Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of ¿ 0.05 mV)
    (c) Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
    (a) Value 1: upsloping
    (b) Value 2: flat
    (c) Value 3: downsloping
12. ca: number of major vessels (0-3) colored by flourosopy
13. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
14. num: diagnosis of heart disease (angiographic disease status)

3.3. Results

4. Conclusion

References

[1] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre-Luc Carrier,

Kyunghyun Cho, Jan Chorowski, Paul Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Mélanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian Goodfellow, Matt Graham, Caglar Gulcehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrancois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol, Olivier Mastropietro, Robert T. McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[2] M.D. Andras Janosi. Heart disease data set. `http://archive.ics.uci.edu/ml/datasets/Heart+Disease`. Hungarian Institute of Cardiology. Budapest.

[3] Yoshua Bengio, Pascal Lamblin, Dan Popovici, Hugo Larochelle, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 19:153, 2007.

[4] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[5] M. Lichman. UCI machine learning repository, 2013.

[6] M.D. Matthias Pfisterer. Heart disease data set. `http://archive.ics.uci.edu/ml/datasets/Heart+Disease`. University Hospital, Basel, Switzerland.

[7] Ph.D. Robert Detrano, M.D. Heart disease data set. `http://archive.ics.uci.edu/ml/datasets/Heart+Disease`. V.A. Medical Center, Long Beach and Cleveland Clinic Foundation.

[8] M.D. William Steinbrunn. Heart disease data set. `http://archive.ics.uci.edu/ml/datasets/Heart+Disease`. University Hospital, Zurich, Switzerland.