Volume 7, Issue 3, March 2017 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A Step Forward Towards Deep Learning

Varshapriya J.
Computer & IT Department,
Veermata Jijabai Technological Institute
Maharashtra, India

Minal Ugale
M.tech Software Engineering,
Veermata Jijabai Technological Institute
Maharashtra, India

Abstract- Deep learning is no more just a buzzword but a fast scenario turning reality. The current market is trying to optimize its processes using machine learning and deep learning. With the trend of acquisition of different AI startups by leading companies, the research in this field has taken a speed of light. Also, the demand for a machine engineer or a deep learning scientist has increased manifold. With this rate, knowing and applying deep learning would not only remain a skill but a necessity. The aim of this survey paper is to make one familiar with the fundamentals of deep learning and how to take a step forward from knowing the buzzword to knowing the subject.

Keywords- deep learning, artificial neural network, deep learning frameworks, cnn, feature engineering

I. INTRODUCTION

Deep learning, which was first theorized in the early 80's, is one paradigm for performing machine learning. And because of a flurry of modern research, deep learning is again on the rise because it's been shown to be quite good at teaching computers to do what our brains can do naturally.[2]

Deep neural networks is widely used in speech recognition, computer vision and natural language processing. DNNs automatically learn suitable features without relying on the experience of the researchers, and then perform the operations.[1]

Deep learning is a relatively new area of machine learning. Where machine learning provides a way to find inexact solutions to computationally difficult tasks, deep learning goes a step further, allowing the computer to build complex concepts from simpler ones.[5] Deep Learning has thus sprung up with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence.[3]

The most commonly cited uses for deep learning are speech recognition, computer vision, and natural language processing. While that may sound like a narrow sphere of operation, many interesting developments have come from real-world implementations of deep learning algorithms.[5]

Example:

Let's say we wanted to program a computer to recognize hand-written digits:

Figure 1 MNIST dataset

Zeros, for instance, are basically one closed loop. But what if the person didn't perfectly close the loop. In this case, we have difficulty differentiating zeroes from sixes. We could establish some sort of cutoff, but how to decide the cutoff in the first place is a big question.

It quickly becomes quite complicated to compile a list of heuristics (i.e., rules and guesses) that accurately classifies handwritten digits. And there are so many more classes of problems that fall into this category.

So instead of trying to write a program, we try to develop an algorithm that a computer can use to look at hundreds or thousands of examples (and the correct answers), and then the computer uses that experience to solve the same problem in new situations. Essentially, the goal is to teach the computer to solve by example, very similar to how we might teach a young child to distinguish a cat from a dog.

II. FEATURE ENGINEERING

Feature engineering helps extracting useful patterns from data so that Machine Learning models can distinguish between classes. For example, there are number of greenish vs. bluish pixels as an indicator of whether a land or water animal is in some picture. This feature is helpful for a machine learning model because it limits the number of classes that need to be considered for a good classification. [4]

Feature Learning

Feature learning is Feature Engineering done automatically by algorithms. In deep learning, convolutional layers are exceptionally good at finding good features in images to the next layer to form a hierarchy of nonlinear features that grow in complexity (e.g. blobs, edges -> noses, eyes, cheeks -> faces). The final layer(s) use all these generated features for classification or regression.

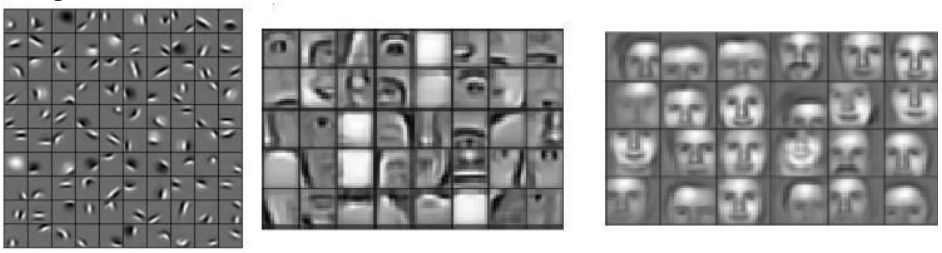

Figure 2 Features generated by a deep learning algorithm

III. DEEP LEARNING

In hierarchical Feature Learning, multiple layers of non-linear features are extracted and passed to a classifier that combines all the features to make predictions. We cannot learn complex features from a few layers. It can be shown mathematically that for images the best features for a single layer are edges and blobs because they contain the most information that we can extract from a single non-linear transformation. To generate features that contain more information we cannot operate on the inputs directly, but we need to transform our first features (edges and blobs) again to get more complex features that contain more information to distinguish between classes.

It has been shown that the human brain does exactly the same thing: The first hierarchy of neurons that receives information in the visual cortex are sensitive to specific edges and blobs while brain regions further down the visual pipeline are sensitive to more complex structures such as faces.

While hierarchical feature learning was used before the field deep learning existed, these architectures suffered from major problems such as the vanishing gradient problem where the gradients became too small to provide a learning signal for very deep layers, thus making these architectures perform poorly when compared to shallow learning algorithms (such as support vector machines).

The term deep learning originated from new methods and strategies designed to generate these deep hierarchies of non-linear features by overcoming the problems with vanishing gradients so that we can train architectures with dozens of layers of non-linear hierarchical features. In the early 2010s, it was shown that combining GPUs with activation functions that offered better gradient flow was sufficient to train deep architectures without major difficulties. From here the interest in deep learning grew steadily.

IV. ARTIFICIAL NEURAL NETWORK

An artificial neural network (1) takes some input data, and (2) transforms this input data by calculating a weighted sum over the inputs and (3) applies a non-linear function to this transformation to calculate an intermediate state. The three steps above constitute what is known as a layer, and the transformative function is often referred to as a unit.

Through repetition of these steps, the artificial neural network learns multiple layers of non-linear features, which it then combines in a final layer to create a prediction. [4]

A standard neural network (NN) consists of many simple, connected processors called neurons, each producing a sequence of real-valued activations. Input neurons get activated through sensors perceiving the environment, other neurons get activated through weighted connections from previously active neurons.[8] The neural network learns by generating an error signal that measures the difference between the predictions of the network and the desired values and then using this error signal to change the weights (or parameters) so that predictions get more accurate.[4]

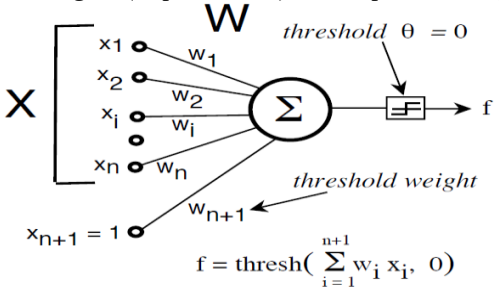

Figure 3 Neuron with Input Vector

Linearly separable (threshold) functions are implemented in a straightforward way by summing the weighted inputs and comparing this sum to a threshold value as shown in the figure. This structure we call a threshold logic unit (TLU).

Its output is 1 or 0 depending on whether or not the weighted sum of its inputs is greater than or equal to a threshold value, theta. The n-dimensional feature or input vector is denoted by X = (x1; : : : ; xn). [6]

1. Unit

A unit often refers to the activation function in a layer by which the inputs are transformed via a nonlinear activation function. Usually, a unit has several incoming connections and several outgoing connections.

2. Activation Function

An activation function takes in weighted data (matrix multiplication between input data and weights) and outputs a non-linear transformation of the data. A unit can have multiple activation functions or a slightly more complex structure.

In deep learning, using non-linear activation functions creates increasingly complex features with every layer. In contrast, the features of 1000 layers of pure linear transformations can be reproduced by a single layer (because a chain of matrix multiplication can always be represented by a single matrix multiplication). This is why non-linear activation functions are so important in deep learning.

3. Layer

A layer is a container that usually receives weighted input, transforms it with a set of mostly non-linear functions and then passes these values as output to the next layer. A layer is usually uniform, that is it only contains one type of activation function, pooling, convolution etc. so that it can be easily compared to other parts of the network. The first and last layers in a network are called input and output layers, respectively, and all layers in between are called hidden layers. [4]

V. CONVOLUTIONAL DEEP LEARNING

Convolution is a mathematical operation which describes a rule of how to mix two functions or pieces of information: (1) The feature map (or input data) and (2) the convolution kernel mix together to form (3) a transformed feature map.

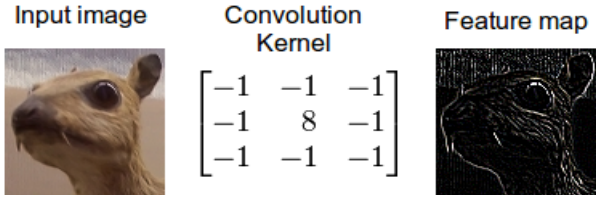

Figure 4 Working of Convolutions

Convolution defines a bridge between the spatial and time domains. This bridge is defined by the use of Fourier transforms: When you use a Fourier transform on both the kernel and the feature map, then the convolution operation is simplified significantly. Some of the fastest GPU implementations of convolutions (for example some implementations in the NVIDIA cuDNN library) currently make use of Fourier transforms.

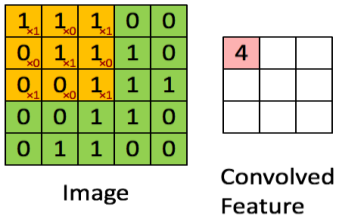

Figure 5 Result of Convolution

While it is unknown which interpretation of convolution is correct for deep learning, the cross-correlation interpretation is currently the most useful: convolutional filters can be interpreted as feature detectors, that is, the input (feature map) is filtered for a certain feature (the kernel) and the output is large if the feature is detected in the image. This is exactly how cross-correlation for an image is interpreted.

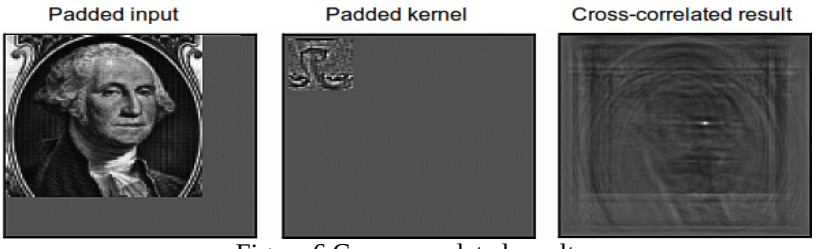

Figure 6 Cross-correlated result

1. Pooling / Subsampling

Pooling is a procedure that takes input over a certain area and reduces that to a single value (subsampling). In convolutional neural networks, this concentration of information has the useful property that outgoing connections usually receive similar information (the information is "funneled" into the right place for the input feature map of the next convolutional layer). This provides basic invariance to rotations and translations.

The larger the size of the pooling area, the more information is condensed, which leads to slim networks that fit more easily into GPU memory. However, if the pooling area is too large, too much information is thrown away and predictive performance decreases. [4]

2. Convolutional Neural Network (CNN)

Convolutional Networks combine three architectural ideas to ensure some degree of shift, scale and distortion invariance: local receptive fields, shared weights and spatial or temporal sub-sampling.[9] A convolutional neural network uses convolutional layers that filter inputs for useful information. These convolutional layers have parameters that are learned so that these filters are adjusted automatically to extract the most useful information for the task at hand.

For example, in a general object recognition task it might be most useful to filter information about the shape of an object (objects usually have very different shapes) while for a bird recognition task it might be more suitable to extract information about the color of the bird (most birds have a similar shape, but different colors; here color is more useful to distinguish between birds). Convolutional networks adjust automatically to find the best feature for these tasks.

Usually, multiple convolutional layers are used that filter images for more and more abstract information after each layer.

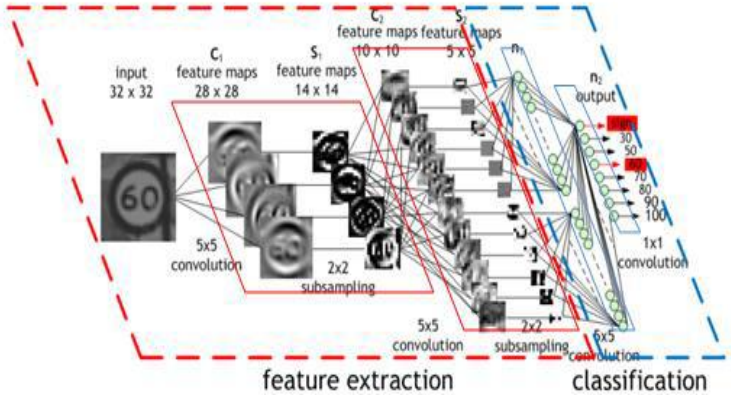

Figure 7 Sample neural net

VI. BACKPROPAGATION OF ERRORS

Backpropagation of errors, or often simply backpropagation, is a method for finding the gradient of the error with respect to weights over a neural network. The gradient signifies how the error of the network changes with changes to the network's weights. The gradient is used to perform gradient descent and thus find a set of weights that minimize the error of the network. [4]

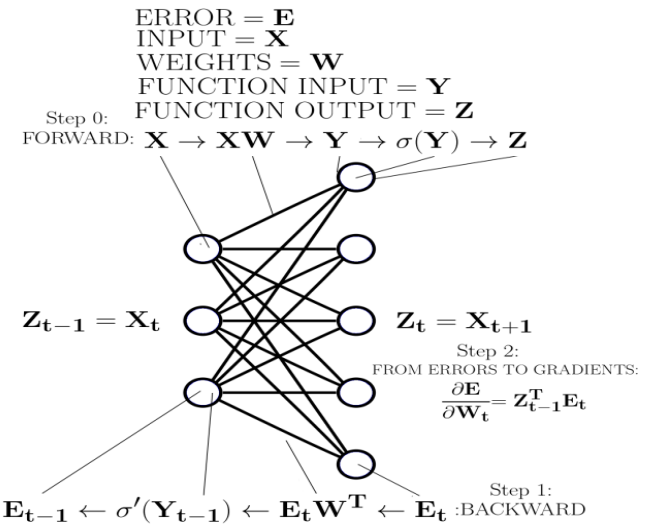

Figure 8 How Backpropagation works

VII. FRAMEWORKS

As the popularity of the deep learning methods have increased over the last few years, several deep learning software frameworks have appeared to enable efficient development and implementation of these methods. The list of available frameworks includes, but is not limited to, Caffe, DeepLearning4J, deepmat, Eblearn, Neon, PyLearn,

TensorFlow, Theano, Torch, etc. Different frameworks try to optimize different aspects of training or deployment of a deep learning algorithm.

Many of these frameworks are mature already as of today and are very fast in training deep networks with billions of parameters thanks to their strong CUDA backends. Today, almost every group training deep networks use Graphical Processing Units (GPU) to accelerate the training process and this has led to joint development of software libraries (e.g. cuDNN) between academic (e.g. Berkeley, NYU) and industry players (e.g. Nvidia).
Some of the important ones can be illustrated as follows:

1. Caffe

Caffe is a deep learning tool developed by the Berkeley Vision and Learning Center and by community contributors and is released under BSD 2-Clause license. It is developed in C++ with expression, speed, and modularity in mind which uses CUDA for GPU computation and has command line, Python, and Matlab interfaces for training and deployment purposes. It separates the definition of the network architecture from actual implementation allowing to conveniently and quickly explore different architectures and layers on either CPU or GPU. Caffe can use LMDB database that allocates memory on the host and device automatically and lazily based on demand for efficient memory usage and high-throughput. The LMDB database supports concurrent reads. Several types of layers and loss functions are already implemented which can be configured in the form of arbitrary directed acyclic graphs in a configuration file. There are also pre-trained models for popular networks such as AlexNet, which allows reproducible research. At the time of writing this report, Caffe supports various layers such as convolution, fully connected and pooling layers, etc. The convolution operation can be computed using either a native implementation (by dense matrix multiplications using Blas) or Nvidia cuDNN, if it is installed, where latter usually results in faster computation. [7]

2. Tensorflow

TensorFlow is a C++ based deep learning framework along with python APIs developed and open sourced under an open source Apache 2.0 License by Google recently. TensorFlow uses data flow graphs for performing numerical computations where the nodes represent mathematical operations and the edges represent multidimensional data array communicated between them. TensorFlow has a flexible architecture that supports multiple backends, CPU or GPU on desktop, server or mobile platforms. TensorFlow also offers users the capability to run each node on a different computational device making it highly flexible. Similar to Theano, TensorFlow has automatic differentiation and parameter sharing capabilities which allows a wide range of architectures to be easily defined and executed. TensorFlow has a fast growing community of users and contributors making it an important deep learning framework within the community. [7]

3. Theano

Theano is a free Python symbolic manipulation library, under a BSD license, aiming to improve execution time and development time for machine learning algorithms. It has specifically been utilized for the gradient-based methods such as deep learning that require repeated computation of the tensor-based mathematical expressions. Such mathematical expressions can be rapidly coded in Theano using a high-level description language similar to a functional language that can be compiled and executed on either a CPU or a GPU. Theano uses CUDA library to define arrays located on an Nvidia GPU memory with Python bindings. In the latest version of Theano (Theano 0.7), the convolution operation automatically uses the optimized Nvidia cuDNN library, if installed, to perform the convolution. It also provides two additional implementations for the convolution operation, an FFT-based implementation and an implementation based on the open-source code of Alex Krizhevsky. While Theano is a general mathematical expression library and may have a relatively steep learning curve for writing efficient code and debugging, several libraries (e.g. Pylearn2, Keras, and Lasagne) have been developed on top it which are specifically tailored for deep learning algorithm providing building blocks for fast experimentation of the well-known methods. [7]

4. Torch

Torch is a scientific computational framework built using Lua that runs on Lua (JIT) compiler. It has strong CUDA and CPU backends and contains well-developed, mature machine learning and optimization packages. The Tensor libraries that come with it have very efficient CUDA backend and the neural networks (nn) libraries can be used to build arbitrary acyclic computation graphs with automatic differentiation functionalities. It has forward() function that computes the output for a given input, flowing the input through the network; and it has backward() function that will differentiate each parameter in the network w.r.t. the gradient that is passed in. Torch also provides bindings to the latest version of Nvidia cuDNN that gives it access to state-of-art speedups for convolutional operations. The latest version, Torch7, has easy to use multi-GPU support and parallelizing packages that make it very powerful for training deep architectures. Torch has a large community of developers and is being actively used within large organizations like Facebook, Google and Twitter. Specifically, many researchers at NYU and Facebook AI Research (FAIR) lab actively contribute to Torch by making a lot of their code open source. Many companies also have in-house teams to customize Torch for their deep learning platforms that has contributed to its popularity in recent times.[7]

VIII. ROLE OF GPUS IN DEEP LEARNING

When it comes to deep learning, GPUs in your architecture provide enormous speed gains. With GPUs, experiments can be done with new algorithms, and the results can be seen much faster – more than 10x faster in many

case. Some algorithms lend themselves to parallelization more than others. For example, convolutional layers can be parallelized fairly easily, and they scale well across multiple GPUs. But GPUs in architecture can be of use without parallelization since one can use them to run multiple algorithms separately, each given its own GPU for processing. GPUs aren't the only options. Accelerators such as Intel's Xeon Phi can also play a role, however, the availability of NVIDIA's CUDA libraries, and their integration into many of the popular deep learning frameworks and libraries makes it easy to start deep learning environment off with NVIDIA GPUs.[5]

IX. RELATED PACKAGES

Beyond the main machine learning libraries and frameworks, there are a number of software components that you should consider. Let's look at a few of those here.

MLpython is a library for organizing machine learning research. Data scientists use it to gain access to many datasets and learning algorithms, and to organize their own research into a simple framework.

CuDNN is NVIDIA's CUDA Deep Neural Network (hence the name cuDNN) library for deep learning. As NVIDIA puts it, "cuDNN
provides highly tuned implementations for standard routines such as forward and backward convolution, pooling, normalization, and activation layers. cuDNN is part of the NVIDIA Deep Learning SDK."

DIGITS is NVIDIA's Deep Learning GPU Training System that lets data scientists quickly design and develop deep neural networks for image classification and object detection. Think of it as a "frontend" for machine learning users.[5]

X. CONCLUSION

The Advent of artificial neural networks ignited a quantum shift from the linear machine learning methods. Neural networks, using multiple hidden layers, have very high capacity to model highly varying functions defining the nonlinear structure of input data. The study of these neural networks has enabled the human to apply it for various use cases in real time.

REFERENCES

[1] Real-time Activity Recognition on Smartphones Using Deep Neural Networks, Licheng Zhang, Xihong Wu and Dingsheng Luo
[2] http://nikhilbuduma.com/2014/12/29/deep-learning-in-a-nutshell/
[3] http://deeplearning.net/
[4] https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-core-concepts/
[5] Building a Deep Learning Environment for your Organization
[6] Introduction to ML, Nils J. Nilsson, Robotics Laboratory, Department of Computer Science, Stanford University
[7] Comparative Study of Deep Learning Software Frameworks, Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, Mohak Shah, Research and Technology Center, Robert Bosch LLC
[8] Deep Learning in Neural Networks: An Overview, Jurgen Schmidhuber
[9] Gradient based learning Applied to Document Recognition, Yann LeCun, Leon Bottou, Yoshua Bengio and Patrick Haffner

