TensorFlow in Deep learning

[1]Mrs.A.Pavithra,[2]Mr.S.Murukanantha Prakash,
[1]*Assistant Professor, Department of Computer Science, Sree Saraswathi Thyagaraja College, Pollachi*
[2]*Student, Department of Computer Science, Sree Saraswathi Thyagaraja College, Pollachi*

Abstract- **Deep learning has revolutionized the technology industry. Modern machine translation, search engines, and computer assistants are all powered by deep learning. TensorFlow is used to do all its complex work very simple. TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google's AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains. This trend will only continue as deep learning expands its reach into robotics, pharmaceuticals, energy, and all other fields of contemporary technology. It is rapidly becoming essential for the modern software professional to develop a working knowledge of the principles of deep learning.**

Index Terms- Deep Learning, TensorFlow, LeNet, AlexNet, ResNet, Logistic Regression.

1. INTRODUCTION

Most deep architectures are built by combining and recombining a limited set of architectural primitives. Such primitives, typically called neural network layers, are the foundational building blocks of deep networks. In this article, we will provide in-depth introductions to such layers. Though, here, we will provide a brief overview of the common modules that are found in many deep networks. It is not meant to provide a thorough introduction to these modules. Rather, we aim to provide a rapid overview of the building blocks of sophisticated deep architectures to whet your appetite. The art of deep learning consists of combining and recombining such modules. We come to know different labels of an image and how it's get reacted with machine learning algorithms.

TENSOR FLOW

Until recently, software engineers went to school to learn a number of basic algorithms (graph search, sorting, database queries, and so on). After school, these engineers would go out into the real world to apply these algorithms to systems. Most of today's digital economy is built on complicated chains of basic algorithms arduously glued together by generations of engineers. Most of these systems are not capable of adapting. All configurations and reconfigurations have to be performed by highly trained engineers, rendering systems brittle.

Machine Learning Eats Computer Science

As the behaviour of software-engineered systems changes, the roles of software engineers will change as well. In some ways, this transformation will be analogous to the transformation following the development of programming languages. The first computers were painstakingly programmed. Networks of wires were connected and interconnected. Then punch cards were set up to enable the creation of new programs without hardware changes to computers. Following the punch card era, the first assembly languages were created. Then higher-level languages like Fortran or Lisp. Succeeding layers of development have created very high-level languages like Python, with difficult ecosystems of precoded algorithms. Much modern computer science even relies on auto generated code. Modern app developers' use tools like Android Studio to auto generate much of the code they'd like to make. Each successive wave of simplification has broadened the scope of computer science by lowering barriers to entry.

2. DIFFERENT LAYERS

2.1 Fully Connected Layer

A fully connected network transforms a list of inputs into a list of outputs. The transformation is called fully connected since any input value can affect any output value. These layers will have many learnable parameters, even for relatively small inputs, but they have the large advantage of assuming no structure in the inputs.

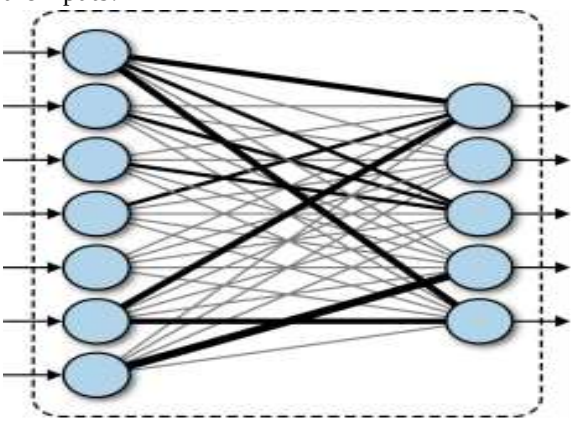

Figure 1: Fully connected Layer

2.2 Convolutional Layer

A convolutional network assumes special spatial structure in its input. In particular, it assumes that inputs that are close to each other spatially are semantically related. This assumption makes most sense for images, since pixels close to one another are likely semantically linked. As a result, convolutional layers have found wide use in deep architectures for image processing.

Just like fully connected layers transform lists to lists, convolutional layers transform images into images. As a result, convolutional layers can be used to perform complex image transformations, such as applying artistic filters to images in photo apps.

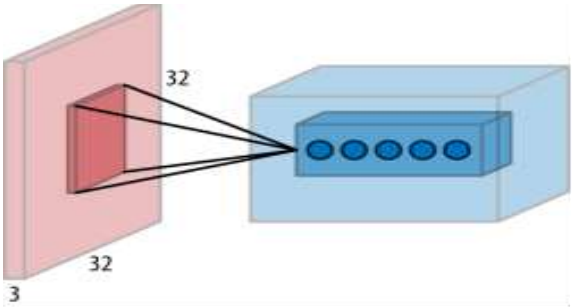

Figure 2: Convolutional Layer

2.3 Recurrent Neural Network Layers

Recurrent neural network (RNN) layers are primitives that allow neural networks to learn from sequences of inputs. This layer assumes that the input evolves from step to step following a defined update rule that can be learned from data. This update rule presents a prediction of the next state in the sequence given all the states that have come previously. An RNN layer can learn this update rule from data. As a result, RNNs are very useful for tasks such as language modelling, where engineers seek to build systems that can predict the next word users will type from history.

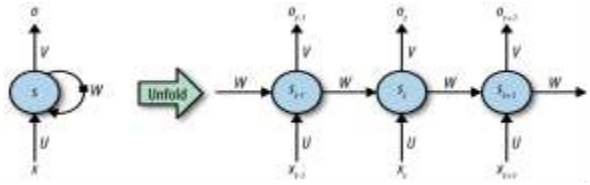

Figure 3: RNN Layer equation

2.4 Long Short-Term Memory Cells

The RNN layers presented in the previous section are capable of learning arbitrary sequence-update rules in theory. In practice, however, such layers are incapable of learning influences from the distant past. Such distant influences are crucial for performing solid language modelling since the meaning of a complex sentence can depend on the relationship between far-away words. The long short-term memory (LSTM) cell is a modification to the RNN layer that allows for signals from deeper in the past to make their way to the present.

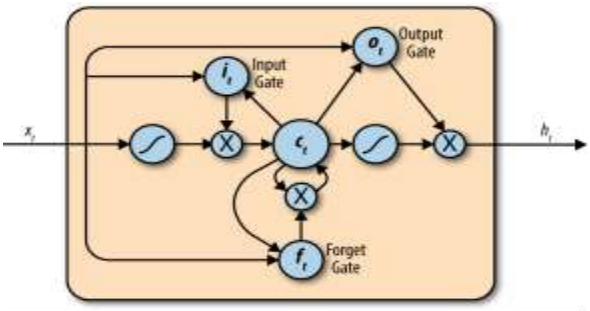

Figure 4: LSTM Layer

3. DEEP LEARNING ARCHITECTURES

There have been hundreds of different deep learning models that combine the deep learning primitives presented in the previous section. Some of these architectures have been historically important. Others were the first presentations of novel designs that influenced perceptions of what deep learning could do. In this article, we present a selection of different

deep learning architectures that have proven influential for the research community.

3.1 Different Architectures

a)LeNet

The LeNet architecture is arguably the first prominent "deep" convolutional architecture. Introduced in 1988, it was used to perform optical character recoginition (OCR) for documents. Although it performed its task admirably, the computational cost of the LeNet was extreme for the computer hardware available at the time, so the design languished in (relative) obscurity for a few decades after its creation.

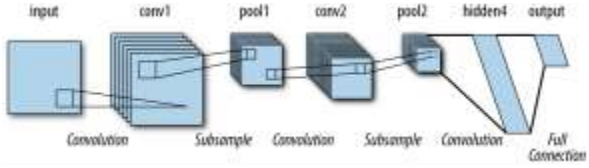

Figure 5: example for LeNet architecture

a)AlexNet

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was first organized in 2010 as a test of the progress made in visual recognition systems. The organizers made use of Amazon Mechanical Turk, an online platform to connect workers to requesters, to catalogue a large collection of images with associated lists of objects present in the image. The use of Mechanical Turk permitted the duration of a collection of data significantly larger than those gathered previously. The first two years the challenge ran, more traditional machine-learned systems that relied on systems like HOG and SIFT features (hand-tuned visual feature extraction methods) triumphed. In 2012, the AlexNet architecture, based on a modification of LeNet run on powerful graphics processing units (GPUs), entered and dominated thechallenge with error rates half that of the nearest competitors. This victory dramatically galvanized the (already nascent) trend toward deep learning architectures in computer vision.

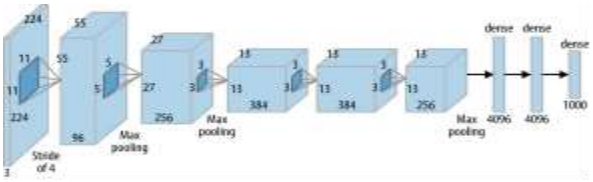

Figure 6: example for AlexNet architecture

b)ResNet

Since 2012, convolutional architectures consistently won the ILSVRC challenge (along with many other computer vision challenges). Each year the contest was held, the winning architecture increased in depth and complexity. The ResNet architecture, winner of the ILSVRC 2015 challenge, was particularly notable; ResNet architectures extended up to 130 layers deep, in contrast to the 8-layer AlexNet architecture. Very deep networks historically were challenging to learn; when networks grow this deep, they run into the vanishing gradients problem. Signals are attenuated as they progress through the network, leading to diminished learning. This attenuation can be explained mathematically, but the effect is that each additional layer multiplicatively reduces the strength of the signal, leading to caps on the effective depth of networks. The ResNet introduced an innovation that controlled this attenuation: the bypass connection. These connections allow part of the signal from deeper layers to pass through undiminished, enabling significantly deeper networks to be trained effectively.

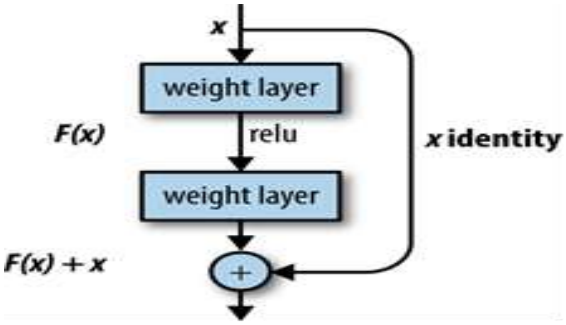

Figure 7: example for ResNet architecture

c)Neural Captioning Model

As practitioners became more comfortable with the use of deep learning primitives, they experimented with mixing and matching primitive modules to create higher order systems that could perform more complex tasks than basic object detection. Neural captioning systems automatically generate captions for the contents of images. They do so by combining a convolutional network, which extracts information from images, with an LSTM layer that generates a descriptive sentence for the image. The entire system is trained end-to-end. That is, the convolutional network and the LSTM network are trained together to achieve the desired goal of generating descriptive sentences for provided images. This end-to-end

training is one of the key innovations powering modern deep learning systems since it lessens the need for complicated pre-processing of inputs. Image captioning models that don't use deep learning would have to use complicated image featurization methods such as SIFT, which can't be trained alongside the caption generator.

d)One-Shot Models

One-shot learning is perhaps the most interesting new idea in machine/deep learning. Most deep learning techniques typically require very large amounts of data to learn meaningful behaviour. The AlexNet architecture, for example, made use of the large ILSVRC dataset to learn a visual object detector. However, much work in cognitive science has indicated that humans can learn complex concepts from just a few examples. Take the example of baby learning about giraffes for the first time. A baby shown a single giraffe at the zoo might be capable of learning to recognize all giraffes she sees from then on. Recent progress in deep learning has started to invent architectures capable of similar learning feats. Such systems can learn to make meaningful predictions with very few data points. One recent paper used this idea to demonstrate that one-shot architectures can learn even in contexts babies can't, such as in medical drug discovery.

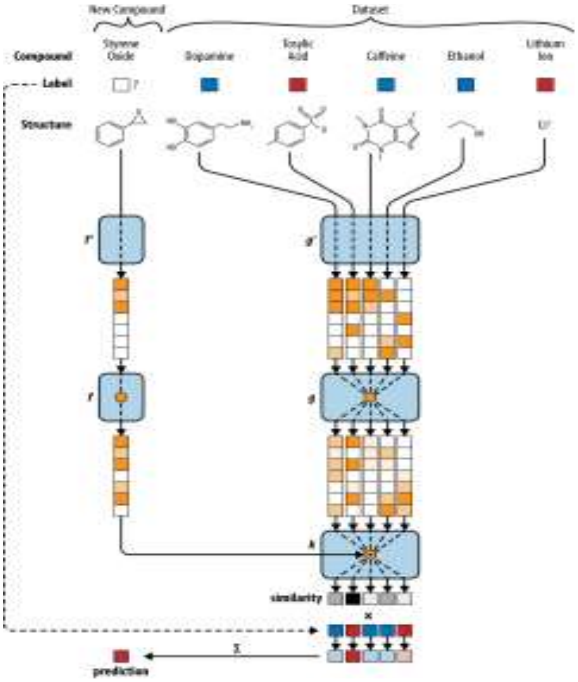

Figure 8: example for One shot

4. DEEP LEARNING FRAMEWORKS

Researchers have been implementing software packages to facilitate the construction of neural network (deep learning) architectures for decades. Until the last few years, these systems were mostly special purpose and only used within an academic group. This lack of standardized, industrial-strength software made it difficult for non-experts to use neural networks extensively. This situation has changed dramatically over the last few years. Google implemented the DistBelief system in 2012 and made use of it to construct and deploy many simpler deep learning architectures. The advent of DistBelief, and similar packages such as Caffe, Theano, Torch, Keras, MxNet, and so on have widely spurred industry adoption.

TensorFlow draws upon this rich intellectual history, and builds upon some of these packages (Theano in particular) for design principles. TensorFlow (and Theano) in particular use the concept of tensors as the fundamental underlying primitive powering deep learning systems. This focus on tensors distinguishes these packages from systems such as DistBelief or Caffe, which don't allow the same flexibility for building sophisticated models.

5. LIMITATIONS OF TENSORFLOW

One of the major current weaknesses of TensorFlow is that constructing a new deep learning architecture is relatively slow (on the order of multiple seconds to initialize architecture). As a result, it's not convenient in TensorFlow to construct some sophisticated deep architectures that change their structure dynamically. One such architecture is the TreeLSTM, which uses syntactic parse trees of English sentences to perform tasks that require understanding of natural language. Since each sentence has a different parse tree, each sentence requires a slightly different architecture.

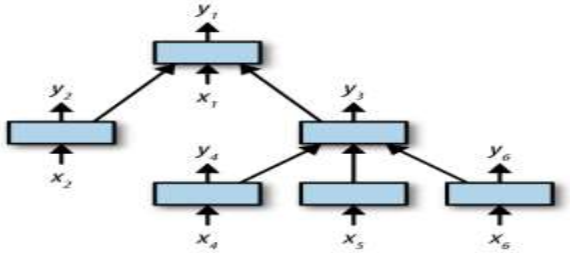

Figure 9: TreeLSTM

While such models can be implemented in TensorFlow, doing so requires significant ingenuity due to the limitations of the current TensorFlow API. New frameworks such as Chainer, DyNet, and PyTorch romise to remove these barriers by making the construction of new architectures lightweight enough so that models like the TreeLSTM can be constructed easily. Luckily, TensorFlow developers are already working on extensions to the base TensorFlow API (such as TensorFlow Eager) that will enable easier construction of dynamic architectures. One takeaway is that progress in deep learning frameworks is rapid, and today's novel system can be tomorrow's old news. However, the fundamental principles of the underlying tensor calculus date back centuries, and will stand readers in good stead regardless of future changes in programming models.

6. TENSORS

We introduced the notion of scalars as rank-0 tensors, vectors as rank-1 tensors, and matrices as rank-2 tensors. What then is a rank-3 tensor? Before passing to a general definition, it can help to think about the commonalities between scalars, vectors, and matrices. Scalars are single numbers. Vectors are lists of numbers. To pick out any particular element of a vector requires knowing its index. Hence, we need one index element into the vector (thus a rank-1 tensor). Matrices are tables of numbers. To pick out any particular element of a matrix requires knowing its row and column. Hence, we need two index elements (thus a rank-2 tensor). It follows naturally that a rank-3 tensor is a set of numbers where there are three required indices.

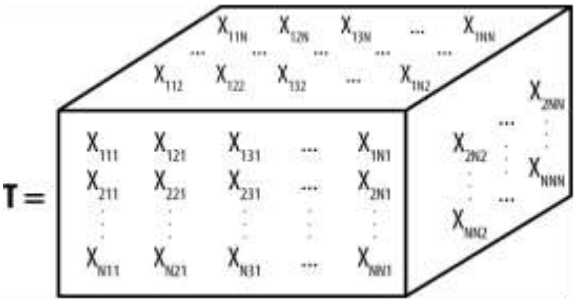

Figure 10: Rank-3 tensor T

The rank-3 tensor T displayed in the figure is of shape (N, N, N). An arbitrary element of the tensor would then be selected by specifying (i, j, k) as indices. There is a linkage between tensors and shapes. A rank-1 tensor has a shape of dimension 1, a rank-2 tensor a shape of dimension 2, and a rank-3 tensor of dimension 3. By our definition, a column vector has shape (n, 1). Wouldn't that make a column vector a rank-2 tensor (or a matrix)? This is exactly what has happened. Recall that a vector which is not specified to be a row vector or column vector has shape (n). When we specify that a vector is a row vector or a column vector, we in fact specify a method of transforming the underlying vector into a matrix. This type of dimension expansion is a common trick in tensor manipulation.

Note that another way of thinking about a rank-3 tensor is as a list of matrices all with the same shape. Suppose that W is a matrix with shape (n, n). Then the tensor Tijk = W1, ⋯,Wn consists of n copies of the matrix W. Note that a black-and-white image can be represented as a rank-2 tensor. Suppose we have a 224 × 224-pixel black and white image.

Then, pixel (i, j) is 1/0 to encode a black/white pixel, respectively. It follows that a black and white image can be represented as a matrix of shape (224, 224). Now, consider a 224 × 224 color image. The color at a particular pixel is typically represented by three separate RGB channels. That is, pixel (i, j) is represented as a tuple of numbers (r, g, b) that encode the amount of red, green, and blue at the pixel, respectively. r, g, b are typically integers from 0 to 255.

It follows now that the color image can be encoded as a rank-3 tensor of shape (224, 224, 3). Continuing the analogy, consider a color video. Suppose that each frame of the video is a 224 × 224 color image. Then a minute of video (at 60 fps) would be a rank-4 tensor of shape (224, 224, 3, 3600). Continuing even further, a collection of 10 such videos would then form a rank-5 tensor of shape (10, 224, 224, 3, 3600). In general, tensors provide for a convenient representation of numeric data. In practice, it's not common to see tensors of higher order than rank-5 tensors, but it's smart to design any tensor software to allow for arbitrary tensors.

7. LOGISTIC REGRESSION

7.1 Metrics for evaluating classification models

Now that you have trained a classification model for logistic regression, you need to learn about metrics

suitable for evaluating classification models. Although the equations for logistic regression are more complicated than they are for linear regression, the basic evaluation metrics are simpler. The classification accuracy simply checks for the fraction of data points that are classified correctly by the learned model. In fact,

With a little more effort, it is possible to back out the separating line learned by the logistic regression model. This line displays the cut-off boundary the model has learned to separate positive and negative examples. We display the learned classes and the separating line in Figure 11. Note that the line neatly separates the positive and negative examples and has perfect accuracy (1.0). This result raises an interesting point. Regression is often a harder problem to solve than classification. There are many possible lines that would neatly separate the data points in Figure 11, but only one that would have perfectly matched the data for the linear regression.

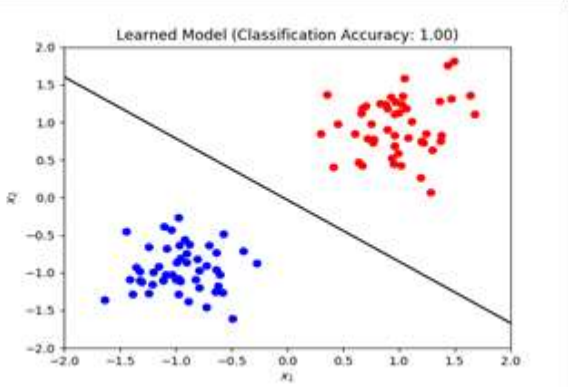

FIGURE 11: VIEWING THE LEARNED CLASSES AND SEPARATING LINE FOR LOGISTIC REGRESSION

8. CONCLUSION

We have shown you how to build and train some simple learning systems in TensorFlow. We started by reviewing some foundational mathematical concepts including loss functions and gradient descent. We then introduced you to some different layers. Artificial intelligence has gone through multiple rounds of boomand- bust development. This cyclical development is characteristic of the field. Each new advance in learning spawns a wave of optimism in which prophets claim that human-level (or superhuman) intelligences are incipient. After a few years, no such intelligences manifest, and disappointed funders pull out. The resulting period is called an AI winter. Then we have discussed with frameworks and limitations of Tensors. Finally, we have discussed about classification in logistic regressions. In future we will implement some hybrid architecture layer for images in different tools.

REFERENCES

[1] Bharat, K.; Broder, A. (1998): A technique for measuring the relative size and overlap of public Web search engines. Computer Networks, 30(1–7), pp. 107–117.

[2] Broder, A.; Kumar, R.; Maghoul, F.; Raghavan, P.; Rajagopalan, S.; Stata, R.; Tomkins, A.; Wiener, J. (2000): Graph structure in the Web. Computer Networks, 33(1–6), pp. 309–320.

[3] Chakrabarti, S. (2000): Data mining for hypertext: A tutorial survey. SIGKDD explorations, 1(2), pp. 1–11.

[4] "A survey on Chromecasat digital device" Journal of Emerging Technologies and Innovative Research (ISSN: 2349-5162) Published in Volume 5 Issue 10, October2018.

[5] "Multimedia and its Applications" International journal for Research and Development in Technology (ISSN: 2349-3585) Published in Volume-10, Issue-5(Nov-18).