

Machine Learning & Neural Networks

Akshit Srivastava

Summer of Science (2019)

Introduction

We come into contact with products based on machine learning everyday in our life without even realizing it. From the famous google page rank algorithm to the applications in financial trading, machine learning is everywhere. Even the algorithm that email service providers use for identifying spam is based on machine learning. Machine learning is almost everywhere. So let's jump to the question of what is machine learning and what is a machine learning algorithm.

According to Arthur Samuel, a pioneer in the field of artificial intelligence and computer gaming,

"Machine learning is a field of Computer Science that gives computers the ability to learn without being explicitly programmed"

Now, let's get to how a Modern Computer Scientist thinks about a Machine Learning Algorithm:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ."

Of course we are no computer scientists, so let's try to put it in a more layman terms. We can say that a machine learning algorithm is a computer program that allows a machine to do get better at doing a job by, if I may say, repeated practice without actually explicitly being programmed to that job. Of course here the practice is the training set we provide the algorithm with (The training set is a collection of relevant data related to the job/task we wish to accomplish). This may sound something like teaching a machine and this is what excites many people. The Big AI dream, though really far away from being realized, can one-day be achieved through Machine Learning techniques. Though we will not talk about AI here, I shall try to touch upon neural networks here, which were our first attempts at trying to emulate the human brain.

Regression and Classification

Now let's get to core machine learning. In machine learning there are two kinds of problems, namely, supervised learning problems and unsupervised learning problems.

Supervised Learning: Supervised learning is the machine learning task of inferring a function from labeled training data.

Unsupervised Learning: Unsupervised learning is the type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses.

The supervised machine learning problem where y is continuous is a **Regression** where as a supervised machine learning problem where y is discrete is a **Classification**.

The most common type of Regression is Linear Regression done by getting a best fit line calculated using Least Squares Method. If you want to go into the mathematics, you can have a look below or if you hate mathematics, just skip!

$$f(a, b) = a + b x,$$

so

$$\begin{aligned} R^2(a, b) &\equiv \sum_{i=1}^n [y_i - (a + b x_i)]^2 \\ \frac{\partial(R^2)}{\partial a} &= -2 \sum_{i=1}^n [y_i - (a + b x_i)] = 0 \\ \frac{\partial(R^2)}{\partial b} &= -2 \sum_{i=1}^n [y_i - (a + b x_i)] x_i = 0. \end{aligned}$$

These lead to the equations

$$\begin{aligned} n a + b \sum_{i=1}^n x_i &= \sum_{i=1}^n y_i \\ a \sum_{i=1}^n x_i + b \sum_{i=1}^n x_i^2 &= \sum_{i=1}^n x_i y_i. \end{aligned}$$

$$b = \frac{\text{cov}(x, y)}{\sigma_x^2}$$

$$a = \bar{y} - b \bar{x}.$$

On the other hand, For the case of a Binary Linear classifier, the Hypothesis function has the following form :

$$z = \mathbf{w}^T \mathbf{x} + b$$
$$y = \begin{cases} 1 & \text{if } z \geq r \\ 0 & \text{if } z < r \end{cases}$$

We are basically saying that it's a function of a linear combination of all the components x_j (hence the word "Linear"). The word 'Binary' refers to the fact that there are only two classes i.e y can only take two discrete values in the problem we are dealing with. And the word 'classifier' refers to the fact that the above function is being used for classifying a given input into one of the two classes present . One of the classes is chosen as a positive class and the other class is chosen as negative class with $y = 0$ corresponding to the negative class and $y=1$ corresponding to the positive class. The function(f) is chosen so that $h(x)$ represents the probability of the input parameters corresponding to the positive class i.e. them corresponding to $y = 1$. One particular case of a Binary Linear Classifier that we are gonna talk about is the Logistic classifier. For a logistic classifier the function(f) is the Logistic Function :

$$y = \frac{a}{1+be^{-rx}}$$

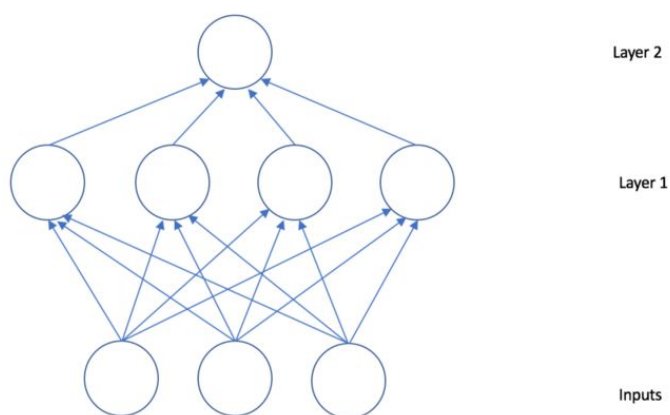
Neural Networks

Enters the subject of my point of interest (in fact for most data scientists these days).

Simply put, a neural network is a connected graph with input neurons, output neurons, and weighted edges. Let's go into detail about some of these components:

- 1) **Neurons**- A neural network is a graph of neurons. A neuron has inputs and outputs. Similarly, a neural network has inputs and outputs. The inputs and outputs of a neural network are represented by input neurons and output neurons. Input neurons have no predecessor neurons, but do have an output. Similarly, an output neuron has no successor neuron, but does have inputs.
- 2) **Connections and Weights**- A neural network consists of connections, each connection transferring the output of a neuron to the input of another neuron. Each connection is assigned a weight.
- 3) **Propagation Function**- The propagation function computes the input of a neuron from the outputs of predecessor neurons. The propagation function is leveraged during the forward propagation stage of training.
- 4) **Learning Rule**- The learning rule is a function that modifies the weights of the connections. This serves to produce a favored output for a given input for the neural network. The learning rule is leveraged during the backward propagation stage of training.

A Deep Neural Network simply has more layers than smaller Neural Networks. A smaller Neural Network might have 1-3 layers of neurons. However, a Deep Neural Network (DNN) has more than a few layers of neurons. A DNN might have 20 or 1,000 layers of neurons.



Hyperparameter Tuning

While our model learns the parameters while training, there are some values for optimizing the rate and accuracy at which these parameters are learnt. Since these control the parameters, they are called hyperparameters and thus this process of searching for the ideal model architecture is referred to as *hyperparameter tuning*.

These hyperparameters might address model design questions such as:


- I. What degree of polynomial features should I use for my linear model?
- II. What should be the maximum depth allowed for my decision tree?
- III. How many neurons should I have in my neural network layer?
- IV. How many layers should I have in my neural network?
- V. What should I set my learning rate to for gradient descent?

Whereas the model parameters specify how to transform the input data into the desired output, the hyperparameters define how our model is actually structured. Unfortunately, there's no way to calculate the way to update my hyperparameter to reduce the loss (ie. gradients) in order to find the optimal model architecture; thus, we generally resort to experimentation to figure out what works best.

In general, this process includes:

1. Define a model
2. Define the range of possible values for all hyperparameters
3. Define a method for sampling hyperparameter values
4. Define an evaluative criteria to judge the model
5. Define a cross-validation method

Grid search is arguably the most basic hyperparameter tuning method. With this technique, we simply build a model for each possible combination of all of the hyperparameter values provided, evaluating each model, and selecting the architecture which produces the best results. Each model would be fit to the training data and evaluated on the validation data. Clearly, this is an exhaustive sampling of the hyperparameter space and can be quite inefficient.



Random Search differs from grid search in that we no longer provide a discrete set of values to explore for each hyperparameter; rather, we provide a statistical distribution for each hyperparameter from which values may be randomly sampled. One of the main theoretical backings to motivate the use of random search in place of grid search is the fact that for most cases, hyperparameters are *not* equally important.

Bayesian optimization belongs to a class of sequential model-based optimization (SMBO) algorithms that allow for one to use the results of our previous iteration to improve our sampling method of the next experiment. The previous two methods performed individual experiments building models with various hyperparameter values and recording the model performance for each. Because each experiment was performed in isolation, it's very easy to parallelize this process. However, because each experiment was performed in isolation, we're not able to use the information from one experiment to improve the next experiment.

Tensorflow

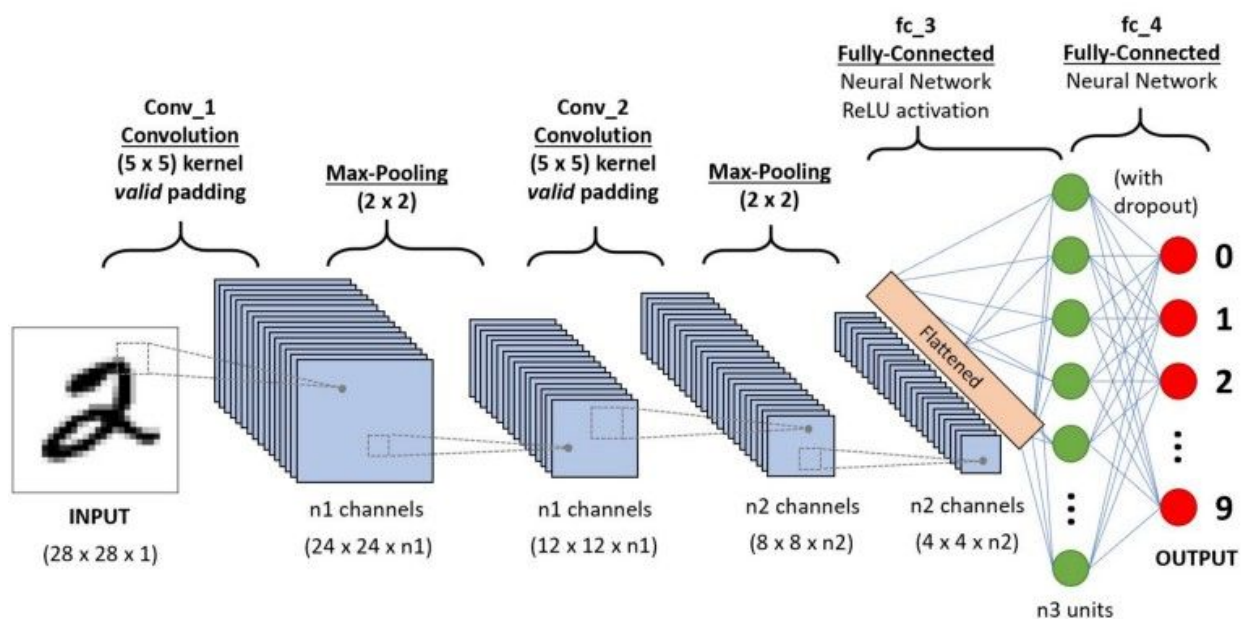
Currently, the most famous deep learning library in the world is Google's TensorFlow. Google product uses machine learning in all of its products to improve the search engine, translation, image captioning or recommendations. It was first made public in late 2015, while the first stable version appeared in 2017. It is open source under Apache Open Source license. You can use it, modify it and redistribute the modified version for a fee without paying anything to Google.

It makes setting up Neural Networks look like a child's play. While I sometimes even struggled to write the code for back-propagation, I just needed less than 10 lines to set up a whole Neural Networks with tensorflow.

Convolutional Neural Networks (CNN)

CNNs are basically a class of deep neural networks, most commonly applied to analyzing visual imagery. In a usual Neural Network, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. However, CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and *assemble more complex patterns using smaller and simpler patterns*. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme. The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.

They generally use a kernel that helps bring out a specific feature (eg - horizontal / vertical edges) followed by Significant reduction of size by a Max-Pooling layer. This is to decrease the computational power required to process the data through dimensionality reduction.

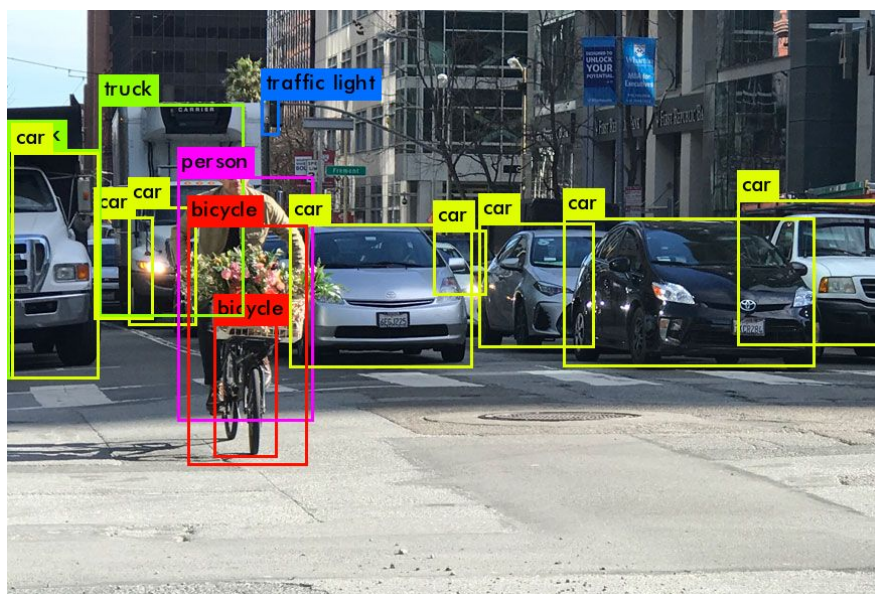


YOLO Algorithm - You Only Look Once

Found in 2016, YOLO is an extremely fast real time multi object detection algorithm. Compared to other region proposal classification networks which perform detection on various region proposals and thus end up performing prediction multiple times for various regions in an image, Yolo architecture is more like Fully CNN and passes the image (nxn) once through the FCNN and output is (mxm) prediction. This architecture is splitting the input image in mxm grid and for each grid generation 2 bounding boxes and class probabilities for those bounding boxes. Arguably, bounding box is more likely to be larger than the grid itself.

A *limitation* of YOLO imposes strong spatial constraints on bounding box predictions since each grid cell only predicts two boxes and can only have one class. This spatial constraint limits the number of nearby objects that our model can predict. It struggles with small objects that appear in groups, such as flocks of birds. A small error in a large box is generally benign but a small error in a small box has a much greater effect on IOU (Interest over Union).

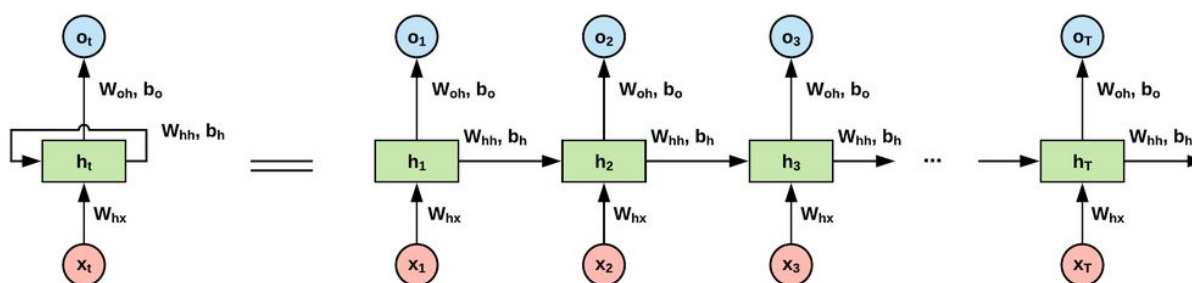
Despite its shortcomings, it is one of the best algorithms for real-time applications.



Sequence Models - RNN

The ability to work with sequence data, like music lyrics, sentence translation, understanding reviews or building chatbots – all this is now possible thanks to sequence modeling, and RNNs are the first step to begin with.

Recurrent Neural Network(RNN) are a type of Neural Network where the **output from previous step are fed as input to the current step**. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is **Hidden state**, which remembers some information about a sequence.



But RNNs themselves have a major problem, error propagate backwards from output to input layer propagating the input error gradient. With deeper neural networks issues can arise from back propagation like vanishing and exploding gradients.

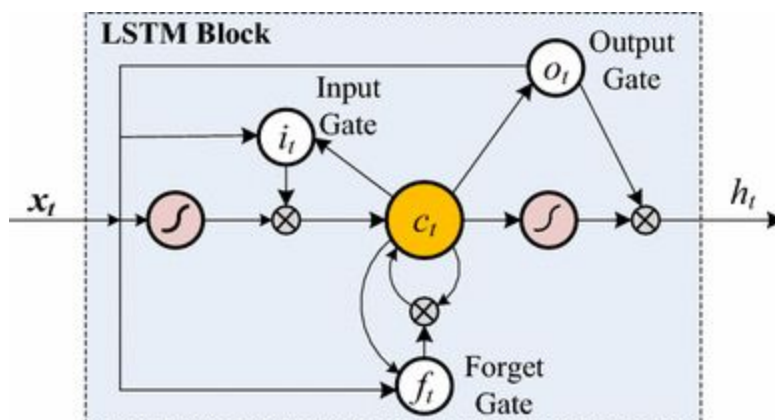
Vanishing Gradients : As we go back to the lower layers. gradient often get smaller. eventually causing weights to never change at lower layers.

Exploding gradients : Opposite of Vanishing gradients, gradient explode on the way back.

This is an issue in Neural Networks but mostly RNN's suffer with Vanishing Gradients due to their large and complex structures.

Luckily, LSTMs come to our rescue.

Long short-term memory is a deep learning system that avoids the vanishing gradient problem. LSTM is normally augmented by recurrent gates called "forget" gates. LSTM prevents backpropagated errors from vanishing or exploding. Instead, errors can flow backwards through unlimited numbers of virtual layers unfolded in space. That is, LSTM can learn tasks that require memories of events that happened thousands or even millions of discrete time steps earlier. LSTM works even given long delays between significant events.



What I learnt in a nutshell :

1. Learnt basics of Machine Learning - Linear Regression, Classification & Clusterization.
2. Learnt about Hyper-parameter tuning and regularisation.
3. Learnt about tensorflow and how it makes implementing neural networks such a bliss.
4. Learnt about Convolutional Neural Networks: ResNets, YOLO Algorithm, Neural Style Transfer.
5. Learnt about Sequence Models: RNNs, Word Embeddings, Beam Search, Attention Models.
6. Implemented all of the above as assignments.

Overview

If you have cared enough to read this and are curious to know how I went about learning all of this, I've jotted down a summary of resources I used with links at appropriate places.

My main motive this summer was to get an insight to the most progressive fields of Computer Science - Deep Learning & Computer Vision via Neural Networks. For that I first watched some NPTEL [lectures](#) to get the gist of all that I am going to cover in the coming days.

Then I did one of the most common specialisations of Coursera - [Deep Learning Specialisation](#).

Further, I completed the first assignment of Stanford's course :[CS-231](#).

While progressing with it, I got a mail from Coursera about a new [course](#) by Deeplearning.ai, that'll help get started with tensorflow, it proclaimed. So I decided to pause the current course and start with the new (and apparently more attractive one).

Even though I mostly used built-in libraries and currently won't consider myself having a thorough knowledge on the mathematics behind all what I did, I did manage to have a glimpse over some of the buzzing topics - Computer Vision, CNNs and RNNs and solve basic problems using them.

I also attempted 2 beginner challenges on [Kaggle](#) to improve my grasp on the topics and learn more about real-life datasets & their cleaning.

Also, [here](#) is the github repository of all the relevant code.

Acknowledgements

First and foremost, I would like to thank my mentor, Ruchika Chavhan, whose enthusiasm, patience and knowledge on this topic motivated me a lot. Getting to learn so much would not have been possible without you guiding the exact resources, trying to answer every little query and helping at every step.

I would also like to thank the Maths and Physics club, IIT Bombay for this wonderful initiative. It helped me learn a lot about a topic I love and helped me spend the long summer vacations in a fruitful way.