Deep Learning

Jorge Diego Hernandez Medina

October 23, 2015

Abstract

This talk gives a brief history of Deep Learning as well as a rough overview of the current techniques and future developments in the field. The talk was written for Nearsoft's October Talks of 2015.

1 Introduction

(Slide 2) Good morning, my name is Jorge Hernandez and I'm a software engineer here at Nearsoft. I learned to program 32 years ago and have been doing it professionally for the last 15 years.

I became a Computer Scientist because I've always been fascinated by the possibility of creating artificial minds. Today I'm going to talk about one of the most promising technologies that we have to make that happen: Deep Learning.

2 What is Deep Learning?

(Slide 4) Representation learning is a set of methods that allow a machine to be fed with raw data and to automatically discover the representations needed for detection or classification. Deep-learning methods are representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules that each transform the representation at one level (starting with the raw input) into a representation at a higher, slightly more abstract level. With the composition of enough such transformations, very complex functions can be learned. For classification tasks, higher layers of representation amplify aspects of the input that are important for discrimination and suppress irrelevant variations.

An image, for example, comes in the form of an array of pixel values, and the learned features in the first layer of representation typically represent the presence or absence of edges at particular orientations and locations in the image. The second layer typically detects motifs by spotting particular arrangements of edges, regardless of small variations in the edge positions. The third layer may assemble motifs into larger combinations that correspond to parts of familiar objects, and subsequent layers would detect objects as combinations of these parts. The key aspect of deep learning is that these layers of features are not designed by human engineers: they are learned from data using a general-purpose learning procedure.

3 Neural Nets

(Slide 6) Frank Rosenblatt was a psychologist at the Cornell Aeronautical Laboratory. He was inspired by the work of Donald O. Hebb, who a decade earlier had predicted how learning might work: As one neuron fires and activates another, repeatedly, the cells improve their joint efficiency. The cells that fire together, wire together, as cognitive scientists like to say. This simple idea, Rosenblatt thought, was enough to build a machine that could learn to identify objects. In 1957 he built it, and called it the Perceptron.

(Slide 7) The Perceptron took up an entire lab room, it worked in three layers. At one end, a square cluster of 400 light sensors simulated a retina; the sensors connected multiple times to an array of 512 electrical triggers, each of which fired, like a neuron, when it passed a certain adjustable threshold of excitement. These triggers then connected to the last layer, which would signal if an object matched what the Perceptron had been trained to see.

(Slide 8) The Perceptron algorithm is a linear classifier.

1. Initialize the weights and the threshold. Weights may be initialized to 0 or to a small random value. In the example below, we use 0.

2. For each example j in our training set D, perform the following steps over the input x_j, and desired output d_j:

a. Calculate the actual output:

$y_j(t) = f[\mathbf{w}(t) \cdot \mathbf{x}_j] = f[w_0(t) + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \cdots + w_n(t)x_{j,n}]$

b. Update the weights:

$w_i(t+1) = w_i(t)+\alpha(d_j - y_j(t))x_{j,i}$ for all $0 \leq i \leq n$, where $0 < \alpha \leq 1$ is the learning rate.

(Slide 9) It was a thrilling development, and Rosenblatt wasnt afraid to share it. In the summer of 1958, he held a news conference with hissponsor, the U.S. Navy. As so often happens in science, he began to talk about the future. To researchers then, he sounded foolish; heard today, prescient. The New York Times caught the gist of it:

The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence. Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language.

Rosenblatts fame irked his peers, many of whom had opted to pursue rules-based artificial intelligence; both sides were chasing after the same military research dollars.

(Slide 10) In 1969 Minsky and Seymour Papert (both at MIT) published *Perceptrons: An Introduction to Computational Geometry* criticizing Rumelhart's connectionist approach.

(Slide 11) The book proved that not only can't a single layer perceptron emulate XOR or XNOR, but also that in three-layered feed-forward perceptrons (with a so-called hidden or intermediary layer), it is not possible to compute some predicates unless at least one of the neurons in the first layer of neurons (the intermediary layer) is connected with a non-null weight to each and every input. Furthermore perceptrons are unable to solve the connectivity problem.

The effect of the book was devastating: virtually no research at all was done in connectionism for 10 years. Eventually, a new generation of researchers would revive the field and thereafter it would become a vital and useful part of artificial intelligence. Rosenblatt would not live to see this, as he died in a boating accident shortly after the book was published.

(Slide 12) In the mid-80s neural networks made a comeback, when David E. Rumelhart and others found a way of teaching more complex functions to a neural net. To accomplish this they designed an algorithm they called backpropagation.

(Slide 13) The backpropagation algorithm looks for the minimum of the error function in weight space using the method of gradient descent:

$$E(\mathbf{w}) = \sum_j (y_j - f_N(\mathbf{x_j}))^2$$

$$\nabla E = (\tfrac{\partial E}{\partial w_1}, \ldots, \tfrac{\partial E}{\partial w_n})$$

Where $f_N(\mathbf{x_j}) = \sigma(\mathbf{x_j})$, $\sigma(x) = \frac{1}{1+e^{-x}}$, and $\frac{\partial E}{\partial w_i}$ is the partial derivative of the error function with respect to the i^{th} weight. For a fixed set of weights ∇E is a vector in $\mathbb{R}^n$ that points in the direction of steepest ascent of the function E.

$$\frac{\partial E}{\partial w_{k,j}} = -\sum_i (y_i - o_i)\frac{\partial o_i}{\partial w_{k,j}}$$

(Slide 14) Where $o_i = f_{N_{1,j}}(\dots) = \sigma(f(\dots))$ represents the value of the impulse function at each of the output neurons. And after applying the chain rule:

$$\frac{\partial E}{\partial w_{k,j}} = -\sum_i (y_i - o_i)\sigma'(f(\dots))w_{j,i}$$

And since $\sigma'(x) = \sigma(x)(1-\sigma(x))$ our update rule (for the weights of N_j) is:

$$\mathbf{w} = \mathbf{w} + \eta o_j(1-o_j)\left(\sum_i w_{j,i}(y_i - o_i)\right)\mathbf{z}$$

Where by $\mathbf{z}$ we denote the vector of inputs to the neuron in question, and η is the learning parameter.

(Slide 15) So then the backpropagation algorithm works as follows:

1. Initialize the weights in the network at random.

2. Evaluate an input $\mathbf{x}$ by feeding it forward through the network and recording at each internal node the output value o_j, and call the final output o.

3. Compute the error for that output value, propagate the error back to each of the nodes feeding into the output node, and update the weights for the output node using our update rule. Repeat this error propagation followed by a weight update for each of the nodes feeding into the output node in the same way, compute the updates for the nodes feeding into those nodes, and so on until the weights of the entire network are updated.

4. Repeat 2 and 3 with a new input $\mathbf{x}$.

Unfortunately the algorithm requires a lot of computation and a large training data set, and when it fails the reasons are not clear. Applying the same algorithm can give different results at different times. This made people choose other methods to work with (like Support Vector Machines) and neural networks faded into obscurity once again.

4 The Return of Neural Networks

(Slide 17) Then in 2006 in a series of papers, Geoffrey Hinton, Yoshua Bengio and Yan LeCun revived interest in what they now called *Deep Learning.*

(Slide 18) Hinton, G. E., Osindero, S. & Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural Comp.* **18**, 15271554

This paper introduced a novel and effective way of training very deep neural networks by pre-training one hidden layer at a time using the unsupervised learning procedure for restricted Boltzmann machines.

Bengio, Y., Lamblin, P., Popovici, D. & Larochelle, H. Greedy layer-wise training of deep networks. In *Proc. Advances in Neural Information Processing Systems 19* 153160

This report demonstrated that the unsupervised pre-training method introduced in the first paper significantly improves performance on test data and generalizes the method to other unsupervised representation learning techniques, such as autoencoders.

Ranzato, M., Poultney, C., Chopra, S. & LeCun, Y. Efficient learning of sparse representations with an energy-based model. In *Proc. Advances in Neural Information Processing Systems 19* 1137 1144

Coupled with the appearance of more generalized GPUs that allowed the computations to be done much faster than before (some nets saw speedups of x30), this led to, for example, Google moving all its Android phones over to use deep learning for speech recognition.

Now I want to tell you about two of the methods being used in deep learning.

5 Convolutional Neural Networks

(Slide 20) Colloquialy called ConvNets, Convolutional Neural Networks are designed to process data that come in the form of multiple arrays, *e.g.* a color image composed of three 2D arrays containing pixel intensities in the RGB channels. Data that comes in the form of multiple arrays are: 1D for language; 2D for images and 3D for video.

There are four key ideas behind ConvNets that take advantage of the properties of natural signals: local connections, shared weights, pooling and the use of many layers.

(Slide 21) As we saw the simplest way to wire a neural network is to connect every to every neuron. But if instead we take advantage of the symmetry in the properties we look for in the data (e.g the frequency of sounds around a given time, or the colors around a particular pixel), we can create a group of neurons, A, that look at small, local segments. That group is called a convolutional layer.

Convolutional layers are often interweaved with pooling layers. For example the max-pooling layer is extremely popular. From a high level perspective, we dont care about precisely where or when a feature is present. A max-pooling layer takes the maximum of features over small blocks of a previous layer. The output tells us if a feature was present in a region of the previous layer, but not exactly where (as if we had zoomed out). They allow later convolutional layers to work on larger sections of the data, and they also make the output invariant to some very small transformations of the data (like noise).

In traditional convolutional layers, A is a bunch of neurons in parallel, that all get the same inputs and compute different features, but in a recent paper *Network In Network* by Min Lin, Qiang Chen and Shuicheng Yan a new Mlpconv layer is proposed. There A would have multiple layers of neurons, with the final layer outputting higher level features for the region.

(Slide 22) What is a convolution? A convolution is a mathematical operation on two functions f and g, producing a third function that is typically viewed as a modified version of one of the original functions.

Imagine we drop a ball from some height onto the ground, where it only has one dimension of motion. How likely is it that a ball will go a distance c if you drop it and then drop it again from above the point at which it landed?

In order to find the total likelihood of the ball reaching a total distance of c, we cant consider only one possible way of reaching c. Instead, we consider all the possible ways of partitioning c into two drops a and b and sum over the probability of each way.

$(f * g)(c) = \sum_{a+b=c} f(a) \cdot f(b)$

6 Recurrent Neural Networks

(Slide 24) Recurrent Neural Networks process an input sequence one element at a time, maintaining in their hidden units a state vector that implicitly contains information about the history of all the past elements of the sequence. We can consider the outputs of the hidden units at different discrete time steps as if they were the outputs of different neurons in a deep multilayer network.

(Slide 25) The artificial neurons (for example, hidden units grouped under node s with values s_t at time t) get inputs from other neurons at previous time steps (this is represented with the black square, representing a delay of one time step, on the left). So the value of s_t is calculated based on the previous hidden state and the input at the current step:

$s_t = f(Ux_t + Ws_{t-1})$

The function f usually is a nonlinearity such as tanh or ReLU (rectified linear unit), and s_{-1}, which is required to calculate the first hidden state, is typically initialized to all zeroes.

In this way, a recurrent neural network can map an input sequence with elements x_t into an output sequence with elements o_t, with each o_t depending on all the previous x_t (for $t \leq t$). For example, if we wanted to predict the next word in a sentence it would be a vector of probabilities across our vocabulary:

$o_t = \text{softmax}(Vs_t)$

The same parameters (matrices U, V, W) are used at each time step. Many other architectures are possible, including a variant in which the network can generate a sequence of outputs (for example, words), each of which is used as inputs for the next time step. The backpropagation algorithm can be directly applied to the computational graph of the unfolded network on the right, to compute the derivative of a total error (for example, the probability of generating the right sequence of outputs) with respect to all the states s_t and all the parameters.

7 The Future of Deep Learning

(Slide 27) Currently why deep learning works so well is not very well understood. Some recent work that hints at why and how has been done using a wavelet scattering transform (which is more amenable to mathematical analysis) using a deep convolution network architecture.

Mark Tygert and others have done work on complex-valued convolutional networks where they have proved that the filter that fits a sequence of inputs is a convolution followed by some mathematical manipulation.

Recently Pankaj Mehta and David Schwab, proved that the statistical technique called renormalization (used by physicists to accurately describe systems without knowing the exact state of all their component parts) also enables neural networks to categorize data as, regardless of transformations such as color, size or orientation.

(Slide 28) Unsupervised learning had a catalytic effect in reviving interest in deep learning, but has since been overshadowed by the successes of purely supervised learning. We expect unsupervised learning to become far more important in the longer term. Human and animal learning is largely unsupervised: we discover the structure of the world by observing it, not by being told the name of every object.

Over the past year, several authors have made different proposals to augment RNNs with a memory module. Proposals include the Neural Turing Machine in which the network is augmented by a tape-like memory that the RNN can choose to read from or write to, and memory networks, in which a regular network is augmented by a kind of associative memory. Memory networks have yielded excellent performance on standard question-answering benchmarks. The memory is used to remember the story about which the network is later asked to answer questions.

Ultimately, major progress in artificial intelligence will come about through systems that combine representation learning with complex reasoning. Although deep learning and simple reasoning have been used for speech and handwriting recognition for a long time, new paradigms are needed to replace rule-based manipulation of symbolic expressions by operations on large vectors.

8 Resources

The following are some resources I found useful while writing this talk.

While it is not an inclusive list I've tried to provide a balance between tutorials and primary sources. Most of the code you'll find for deep learning is in Python.

1. `http://jmozah.github.io/links/`
2. `http://colah.github.io/`
3. `http://www.thetalkingmachines.com/blog/`
4. `http://www.cs.toronto.edu/~hinton/deeprefs.html`
5. `http://www.docdroid.net/11p1b/hinton.pdf.html`
6. `http://deeplearning.stanford.edu/tutorial/`
7. `http://deeplearning.net/tutorial/contents.html`
8. `http://jeremykun.com/2012/12/09/neural-networks-and-backpropagation/`
9. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`
10. `http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/`
11. `https://docs.google.com/file/d/0BxKBnD5y2M8NVHRiVXBnOVpiYUk/edit`
12. `https://drive.google.com/file/d/0BxKBnD5y2M8NbWN6XzM5UXkwNDA/view?pli=1`
13. `https://www.youtube.com/watch?v=EK61htlw8hY`
14. `http://blog.shakirm.com/wp-content/uploads/2015/10/Bayes_Deep.pdf`
15. `http://arxiv.org/abs/1312.4400`
16. `http://arxiv.org/pdf/1503.03438v1.pdf`
17. `http://arxiv.org/pdf/1203.1513.pdf`
18. `http://arxiv.org/abs/1410.3831`
19. `https://www.quantamagazine.org/20141204-a-common-logic-to-seeing-cats-and-cosmos/`
20. `https://charlesmartin14.wordpress.com/2015/04/01/why-deep-learning-works-ii-the-renormalization-group/`

9 Closing

(Slide 30) Does anyone have any questions? Q&A

(Slide 31) I'd like to thank everyone of you for your time and for listening to me ramble on.