Deep Learning

Tom Runia
University of Amsterdam
QUVA Deep Vision Lab
runia@uva.nl

Abstract

Artificial intelligence seems to be present everywhere in the technology industry. Over the last couple of years, we have seen impressive breakthroughs in artificial intelligence: the rise of self-driving cars, the animal-like robots of Boston Dynamics, exciting image recognition systems, and DeepMind's AlphaGo winning from the world champion in the game of Go. Increasingly these systems are powered by a technique called Deep Learning. This branch of machine learning utilizes deep neural networks for building complex feature representations and learning highly non-linear functions for solving classification or regression problems. In this article, we present the reader with an introduction to neural networks and discuss some fascinating applications empowered by deep learning.

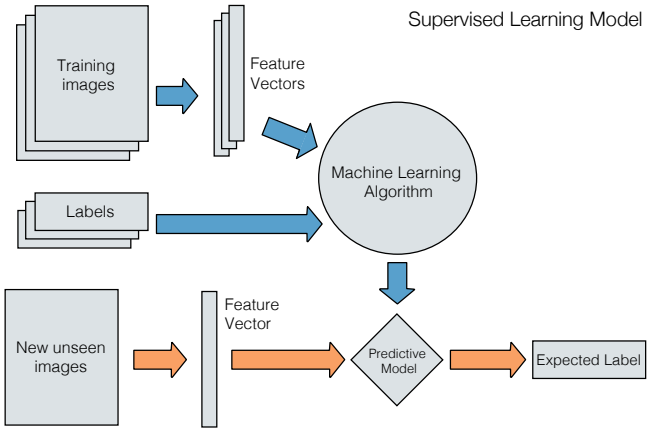

Figure 1: Example of supervised learning. Training a model for image *classification* by learning from a large amount of labeled training data.

1. Introduction

Over the last couple of years, it is hard to have missed the uprise in artificial intelligence (AI) in the news. Scientists all over the world have devoted an incredible amount research to the field, leading to important breakthroughs which enable a wide range of new technological possibilities. In the world of *big data* that we live in, arguably the most promising approach to building smart systems is the branch of AI called *machine learning*. The main focus of machine learning is the development of algorithms that can learn models from large amounts of data, and use these models to make predictions based on new unseen data. Such models make data-driven predictions instead of following strictly static program instructions, hence making them highly relevant in real-world applications. Machine learning is driving technology behind many modern systems including identifying objects in images, self-driving cars, voice recognition, language translation and recommendation systems. More and more, these systems are using deep *neural networks* for representation learning and constructing highly non-linear functions for problem-solving. This renewed interest in neural networks is commonly known as *deep learning*.

Before discussing the power of deep learning, we look at the classic machine learning problem of *image classification*. Given an input image, the task is to predict the class (label) of the visual content, *e.g.* birds, cars or humans. The conventional supervised learning approach to this problem is training a model from a large amount of training data from which we extract certain feature vectors (Figure 1). These features typically give a numerical representation of the local image regions in terms of shape, color or texture. Computing these local descriptors requires a powerful feature extractor that transforms the input image into a suitable internal representation from which the classifier could classify input images. For

that reason, when deploying machine learning in practice, much of the actual engineering effort is put into the data preprocessing and feature extraction pipeline (Bengio *et al.* , 2013). Feature engineering has long been a thorn in the side of machine learning: ideally, we would like to automatically find the best features for a given problem. Deep learning is a family of machine learning algorithms that alleviates the feature engineering by proving powerful *representation learning*.

Representation learning is a set of methods that can automatically discover internal representations by feeding raw data into the system. The most popular type of representation learning is deep learning, in which multiple levels of increasingly complex representations are built from raw input data. Progressing through the pipeline, at each level simpler features are combined using a series *non-linear* modules that each transform the representation at one level (LeCun *et al.* , 2015). By stacking a large amount of such non-linearities, very complex functions can be learned. These complex functions are well-suited for performing classification or regression in challenging tasks such as the image classification problem. Learning complex functions sounds formidable, but how do we devise these algorithms? In the next section, we discuss the ideas behind neural networks which are nowadays the most widely used technique for these learning systems.

2. Neural Networks

Neural networks (Rosenblatt, 1958) originate from the 1950s but were very hard to train back in the days due to limited amounts of training data and insufficient computing resources. Currently, neural networks are one of the most active research areas in computer science. Better training techniques (*e.g.* dropout), powerful graphic processing units (GPUs) for large matrix multiplications and enormous training datasets have made it possible to train effectively and obtain state of the art performance on many benchmarks.

Neural networks are models inspired by biological networks in our brain and are generally structured as a set of interconnected neurons (or units) that can communicate with each other. Figure 2 displays a neural network organized in layers; one input layer followed by two *hidden* layers and a final output layer. The remainder of this section discusses the process of

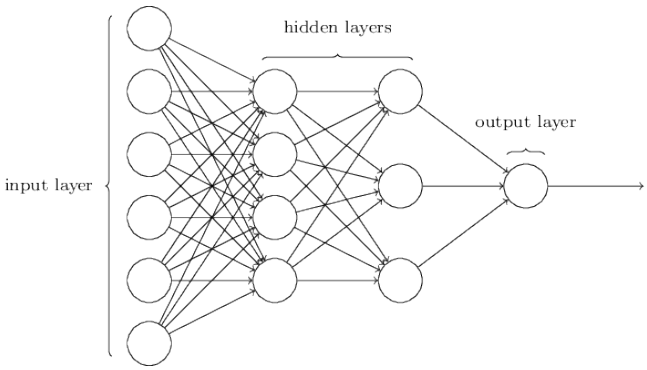

Figure 2: Example architecture of a neural network with two hidden layers. The input can be raw vectorized data and the output corresponds to a vector of classification scores, one for each object class. Image is taken from (Nielsen, 2016).

training a neural network in the supervised learning context for performing a classification task.

Again, looking at the example of image classification, the input layers can be fed with raw pixels from images x and the desired output $y = y(x)$ is a one-hot vector encoding the object class. For example, if we have a problem with 6 classes and object x_i belongs to class 2 we denote $y(x_i) = (0, 1, 0, 0, 0, 0)^T$. During training, the network is presented a large amount of training images with ground truth labels. The core idea behind learning is that we define a *loss function* $\mathcal{L}(w, b)$ that measures the error between the ground truth and predictions of the network. This loss function is expressed in terms of the adjustable variables: weights w and biases b of our network corresponding to the edges and nodes respectively of the network in Figure 2. The quadratic loss function is a simple example (although rarely used in practice):

$$\mathcal{L}(w, b) = \frac{1}{2n} \sum_{i=0}^{n} || \, y(x_i) - a \, ||^2 \qquad (1)$$

Where n is the number of training examples, and a is the vector of output scores from the network when x is the input. By minimizing the loss function we hope to find network weights that are well-suited for discriminating between all object classes. One might ask, why not directly optimize the number of images correctly classified by the network instead of the loss function? The problem lies in the fact that the number of correctly classified images is not a smooth function of the weights and biases. For this reason, it is difficult to find out the effect of small changes in weights and

biases on the performance of the network (Nielsen, 2016). By iteratively feeding the machine with training examples, the network modifies its internal parameters to reduce the loss. State of the art networks for image classification and speech recognition can have hundreds of millions of these adjustable weights and several millions of labeled examples are required for learning these parameters.

The question that remains is how to adjust the "knobs" w and b in each iteration to minimize the loss function. Adjusting the weights and biases for minimizing the loss function can be done by computing the gradient vector for each weight, or in other words, the increase or decrease in loss as a result of a small modification in the weight. The weight vector is then adjusted in the opposite direction to the gradient vector. As long as the internal units (neurons) are relatively smooth functions[1] the *backpropagation* procedure (Rumelhart *et al.* , 1988) can be used as practical application of the chain rule for derivatives. Starting from the last (prediction) layer of the network, the backpropagation procedure can be applied repeatedly to propagate the gradients through all units in the network until the input layer is reached. Using the gradients to update the weights and biases to minimize the loss is then straightforward. The discovery of backpropagation in the 1980s was a major breakthrough in machine learning and currently is the workhorse of training deep neural networks because it is efficient in for propagating gradients through neural networks with many hidden layers, *i.e.* deep neural networks.

In practice each training iteration performs a forward and backward pass through the network. For a batch of labeled examples, the network output is evaluated in the forward pass. Then the loss is computed and the gradients are back-propagated through the network. Upon the gradients reaching the input layer, the gradients are used for updating the weights and biases of the network. This process continues for a number of training *epochs* (an epoch consists of processing all the training examples once). Training deep networks with many layers requires extremely large datasets and can easily take weeks of training time. To put this in context, in our lab we are running a cluster of NVIDIA Titan X GPUs (each with 3072 CUDA cores), and even then, training deep networks can take more than a week on the well-known ImageNet dataset[2].

On a high-level, in this section we have seen how neural networks can be utilized for finding complex representations from raw data. This is done by iteratively feeding training examples to the network and updating the weights and biases depending on the loss of the network. Important to notice is that the weights correspond to the representation that is automatically learned from the raw data. With this we conclude our brief introduction to neural networks. Although many important theoretical and practical considerations are not discussed, the high-level ideas in this section form the core of deep learning and are applied everywhere in modern machine learning. At the end of this article we refer the interested reader to additional reading on training deep neural networks. We continue by reviewing a number of interesting applications made possible by deep learning.

3. Applications of Deep Learning

The ability to train very large networks that are capable of learning very complex representations by progressively combining simple features using non-linear transformations has led to a large amount of practical applications and intriguing experiments. In this section, we will have a look at some possibilities powered by deep learning.

3.1. Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (LeCun *et al.* , 1989) are currently the most popular deep learning architecture, well-suited for understanding *images*. Loosely inspired by the layers in the visual cortex of primate brains, CNNs consists of increasingly complex convolutional filter layers that are learned during the training process. The classical CNN consist of alternatively stacked convolutional layers and spatial pooling layers: each convolution layer generate a feature map by linear convolutional filters followed by a non-linear activation function (Figure 4). Thus, each convolution layer progressively extracts higher-level features, until the final layer essentially makes a decision on what the image shows. The adjustable parameters of the network are the weights and biases of the convolutional filters, *i.e.* the networks automatically learns a set of feature maps that are highly descriptive for the images.

[1] In practice, sigmoid functions $f(x) = \frac{1}{1+e^{-x}}$ and rectified linear units (ReLU) $f(x) = \max(0, x)$ are commonly used as non-linear activations functions for a neuron.

[2] http://www.image-net.org/challenges/LSVRC/

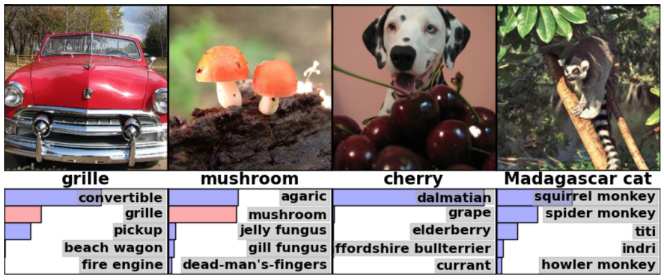

Figure 3: Top-5 class predictions of ImageNet test images using `AlexNet` ConvNet architecture. Image is taken from (Krizhevsky *et al.* , 2012).

CNNs demonstrate the power of deep learning very well. For the image classification task on the ImageNet benchmark, which requires recognizing $1,000$ object classes, state-of-the-art CNNs with 150 layers achieve an error rate of 3.5%. The object classes are highly complex and require a great level of specific knowledge, some examples are visualized in Figure 3. Trained humans achieve a performance of approximately 5% error rate (Karpathy, 2015b), clearly displaying the unbelievable discriminative power of deep learning.

Particularly interesting for ConvNets is that the learned weights can be visualized as images. This experiment was first performed by Zeiler and Fergus (2014), see Figure 5. These feature map visualizations show how the network learns increasingly complex features. The first layer extracts very simple features such as edges or corners. Intermediate layers construct higher-level features that respond to blobs, certain texture and increasingly complex shapes progressing through the layers. It is intriguing to observe how computers automatically learn visual features in the first layers (*e.g.* edges, corners) similar to visual representations that we humans have learned in the first layers of our visual cortex (DiCarlo *et al.* , 2012).

3.2. Recurrent Neural Networks (RNNs)

Traditional neural networks have the major shortcoming of not being able to deal with sequential data, such as speech and language. Recurrent neural networks are a very popular family of deep learning architectures that tackle this problem, by storing information about previous events internally. RNNs process an input sequence one element at a time, maintaining an internal representation of the history of past elements of the sequence. More concretely, in traditional neural networks the behavior of hidden neurons is only determined by previous hidden layers, while for RNNs the weights are also determined by the activations at earlier times. While many types of RNNs architectures are described in the literature, the most popular are the Long Short-Term Memory (LSTM) networks, capable of learning long-term dependencies (Hochreiter & Schmidhuber, 1997).

One example showing the success of RNNs is the voice recognition software on your smartphone. More interestingly however is the fact that RNNs can also be used for *generating* new sequential data such as text. RNNs can be trained on large amounts of text to generate new text of the same kind. Experiments show that these systems are already capable of writing simple stories that are grammatically correct (for example by training on the works of Shakespeare or Wikipedia). My favorite example is training a recurrent network on large amounts of LaTeXsource. After several hours of training the machine begins to understand the syntactic language and is able to generate "hallucinated" algebraic geometry and other types of mathematics (Karpathy, 2015a). Figure 6 displays an example. Although the generated mathematics is meaningless, it is very interesting that the machine is able to learn how write text with few mistakes. For the same research, the authors also trained LSTMs on the complex Linux source code, also with surprisingly well-written code produced by the machine that compiled right away.

4. Conclusion

In this article, we have presented the reader with an introduction to neural networks and a primer for some fascinating applications of deep learning. What else is going on in neural networks and deep learning? There is a huge amount of fascinating work going on. Active research areas include unsupervised learning, reinforcement learning, language processing and video understanding. Many experts argue that unsupervised learning is becoming increasingly important for machine learning. Obtaining enormous datasets of *labeled* training examples is a very costly process while unlabeled data is inexhaustible. With this regard, we end with a quote of Yann LeCun, one of the founding fathers of deep learning: "*We need to solve the unsupervised learning problem before we can even think of getting to true AI. And that's just one obstacle we know about. What about all the ones we don't know about?*"

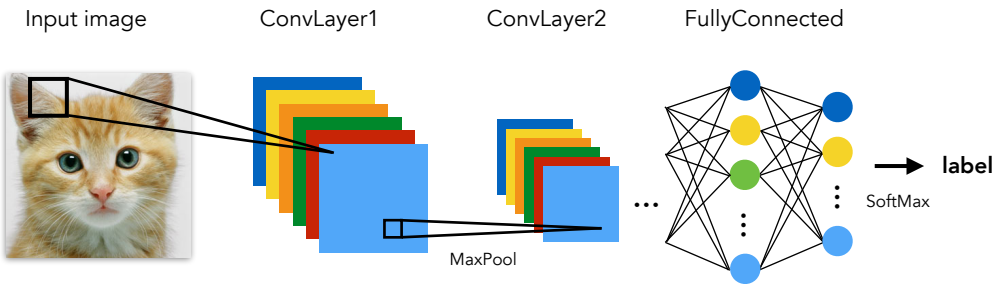

Figure 4: Example of a convolutional neural network (CNN) architecture. For classification the last fully-connected layer produces a predicted probability distribution over all object classes. State-of-the-art CNN architectures for image classification can have more than 100 layers.

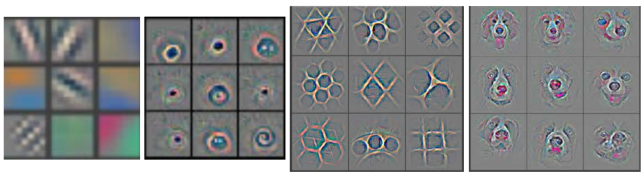

Figure 5: Visualization of feature maps learned in first four convolutional layers by training on millions of images. Starting from the first layer (left), we notice that the learned features become increasingly complex moving to higher layers. Image is taken from (Zeiler & Fergus, 2014).

4.1. Further Reading

In particular over the last three years, an astonishing amount of literature has been written on deep learning. The literature in the current generation of deep learning is all very recent and good (published) books are non-existent. However, two books in preparation for press are "Deep Learning" (Ian Goodfellow & Courville, 2016) and "Neural Networks and Deep Learning" (Nielsen, 2016). Additionally, reading all the papers by the founding fathers of deep learning: Geoffrey Hinton, Yann LeCun and Yoshua Bengio is highly recommended.

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*
Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma **??**. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

Proof. Let $\mathcal{F}$ be a sheaf of $\mathcal{O}$-modules. Let $\mathcal{F}$ be a category.

1

Figure 6: Hallucinated mathematics generated by a multilayer LSTM trained on a large amount of LaTeXsource. Note how the proof is omitted by the machine (haha). Image is taken from (Karpathy, 2015a).

References

Bengio, Yoshua, Courville, Aaron, & Vincent, Pierre. 2013. Representation learning: A review and new perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **35**(8), 1798–1828.

DiCarlo, James J, Zoccolan, Davide, & Rust, Nicole C. 2012. How does the brain solve visual object recognition? *Neuron*, **73**(3), 415–434.

Gatys, Leon A., Ecker, Alexander S., & Bethge, Matthias. 2015. A Neural Algorithm of Artistic Style. *CoRR*, **abs/1508.06576**.

Hochreiter, Sepp, & Schmidhuber, Jürgen. 1997. Long short-term memory. *Neural computation*, **9**(8), 1735–1780.

Ian Goodfellow, Yoshua Bengio, & Courville, Aaron. 2016. *Deep Learning*. Book in preparation for MIT Press.

Karpathy, Andrej. 2015a. *The Unreasonable Effectiveness of Recurrent Neural Networks*.

Karpathy, Andrej. 2015b. *What I learned from competing against a convnet on ImageNet*.

Krizhevsky, Alex, Sutskever, Ilya, & Hinton, Geoffrey E. 2012. Imagenet classification with deep convolutional neural networks. *Pages 1097–1105 of: Advances in neural information processing systems*.

LeCun, Yann, Boser, Bernhard, Denker, John S, Henderson, Donnie, Howard, Richard E, Hubbard, Wayne, & Jackel, Lawrence D. 1989. Backpropagation applied to handwritten zip code recognition. *Neural computation*, **1**(4), 541–551.

LeCun, Yann, Bengio, Yoshua, & Hinton, Geoffrey. 2015. Deep learning. *Nature*, **521**(7553), 436–444.

Nielsen, Micheal. 2016. *Neural Networks and Deep Learning*. Determination Press.

Rosenblatt, Frank. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, **65**(6), 386.

Rumelhart, David E, Hinton, Geoffrey E, & Williams, Ronald J. 1988. Learning representations by back-propagating errors. *Cognitive modeling*, **5**(3), 1.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael S., Berg, Alexander C., & Li, Fei-Fei. 2014. ImageNet Large Scale Visual Recognition Challenge. *CoRR*, **abs/1409.0575**.

Zeiler, Matthew D, & Fergus, Rob. 2014. Visualizing and understanding convolutional networks. *Pages 818–833 of: Computer vision–ECCV 2014*. Springer.