UNIVERSITAT POLITÈCNICA DE CATALUNYA
UNIVERSITAT DE BARCELONA
UNIVERSITAT ROVIRA I VIRGILI

Master in Artificial Intelligence

Master of Science Thesis

MY MASTER THESIS TITLE

Hadi Keivan Ekbatani

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
FACULTAT DE MATEMÀTIQUES (UB)
ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA (URV)

Supervisor:
Oriol Pujol Vila
Department of analysis
and applied Mathematics,
Universitat de Barcelona (UB)

Co-supervisor:
Santiago Segui Mesquida
Department of analysis
and applied Mathematics,
Universitat de Barcelona (UB)

February 01, 2016

Acknowledgments

I would like to sincerely thank my supervisors Oriol Pujol and Santi Segui for their support, guidance and mentorship. I greatly appreciate their demanding and inquisitive scientific attitude, while keeping always a calm and positive mindset. They are, in my mind, an inspiring example of what university professors should be.

Furthermore I would also like to dedicate this master thesis to my beloved parents and sister for their unsparing supports. My gratitude knows no bounds.

Another special acknowledgment must be made to my friends in the master: Denis, Jeroni, Philipp, Pablo, Lorenzo, Iosu, Ferran and many others for the endless studying hours, morning coffees after crunching the brutal assignments all night long, valuable and constructive discussions which allowed me carry out this program.

Abstract

NOT YET

Contents

List of Figures

1 Introduction

Learning to count is an important educational/developmental milestone which constitutes the most fundamental idea of mathematics. In Computer Vision, the counting problem is the estimation of the number of objects in a still image or video frame. Learning to count visual objects is a new approach towards dealing with detecting objects in the images and video has been recently proffered in the literature. It arises in many real-world applications including cell counting in microscopic images, monitoring crowds in surveillance systems, and performing wildlife census or counting the number of trees in an aerial image of a forest[13].

1.1 Motivations

1.2 Objectives

1.3 Contributions of the Research

1.4 Organization

2 Background and definitions

In this section, we go over the preliminarily concepts that help understand the contributions of this thesis. We start by looking at the family of methods to which Deep Convolutional Neural Networks belong to, followed by a more detailed look at the method in question and explain the some hyper-parameters incorporated for optimizing the proposed model.

2.1 Deep Learning

The true challenge to Artificial Intelligence(AI) proved to be solving the tasks that are easy for people to perform but hard for people to describe formally—problems that we solve intuitively, that feel automatic, like recognizing spoken words or faces in images. The solution is to allow computers to learn from experience and understand the world in terms of a hierarchy of concepts, with each concept defined in terms of its relation to simpler concepts. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones. If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI *Deep Learning*[5].

Modern deep learning provides a very powerful framework for supervised learning. By adding more layers and more units within a layer, a deep network can represent functions of increasing complexity. Most tasks that consist of mapping an input vector to an output vector, and that are easy for a person to do rapidly, can be accomplished via deep learning, given sufficiently large models and sufficiently large datasets of labeled training examples. Other tasks, that can not be described as associating one vector to another, or that are difficult enough that a person would require time to think and reflect in order to accomplish the task, remain beyond the scope of deep learning for now[5].

In other words, Deep Learning is a new area of Machine Learning research, which has been introduced with the objective of moving ML closer to one of its original goals: AI. Deep Learning is about learning multiple levels of representation and abstraction that help to make sense of data such as images, sound and text[19].

2.2 Deep Neural Networks

A standard neural network (NN) consists of many simple, connected processors called neurons, each producing a sequence of real-valued activations. Shallow NN-like models with few such stages have been around for many decades if not centuries[17]. However, theoretical results strongly suggest that in order to learn the kind of complicated functions that can represent high-level abstractions (e.g. in vision, language, and other AI-level tasks), one needs deep architectures. Deep Neural Networks are composed of multiple levels of non-linear operations, such as in neural nets with many hidden layers or in complicated propositional formulate re-using many sub-formulate[1].

2.2.1 Back Propagation

Backward Propagation of errors (BP) was the main advance in the 1980's that led to an explosion of interest in NNs. BP is one of the most common used methods for training NNs. The idea behind BP is that it repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the network and the desired one. As a result of the weight adjustments, internal *hidden* units come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units[22].

In other words, BP computes how fast the error changes as we change a hidden activity by using error derivatives with respect to hidden activities. Since each hidden activity can have a notable effect on many output units and consequently on the error, a combination these effects must be considered. This combination is done efficiently which allows us to compute error derivatives for all the hidden units efficiently at the same time. Computing the error derivatives for the hidden activities, it would be easy to get the error derivatives for the weights going into a hidden unit which is the key to be able to learn efficiently.

2.2.2 Weight Sharing

Weight sharing consists in having several connections be controlled by a single parameter (weight). Weight sharing can be interpreted as imposing equality constraints among the connection strengths. An interesting feature of weight sharing is that it can be implemented with very little computational overhead[10]. The weight sharing technique has the interesting side effect of reducing the number of free parameters, thereby the capacity of the machine and improving its generalization ability[12].

2.3 Convolutional Neural Networks

Convolutional Neural Networks are a specialized kind of neural network for processing data that has a known grid-like topology such as image data which can be thought of as a 2D grid of pixels. CNN are simply neural networks that use convolution in place of general matrix multiplication in at least on of their layers[5]. Basically , CNN combine three architectural ideas to ensure some degree of shift and distortion invariance local receptive fields, shared weights (or weight replication), and sometimes, spatial or temporal sub-sampling[12]. In the main body of any CNN's architecture, the following components exist:

2.3.1 Convolutional layer

Each unit of a convolutional layer receives inputs from a set of units located in a small neighborhood in the previous layer. With local receptive fields, neurons can extract elementary visual features such as oriented edges, end-points and corners. these features are then combined by the higher layers[12]. In addition, elementary feature detectors that are useful on one part of the image are likely to be useful across the entire image. This knowledge can be applied by forcing a set of units, whose receptive fields are located at different places on the image, to have identical weight vectors[22]. The outputs of such a set of neurons constitute a *feature map*. At each position, different types of units in different feature maps compute different types of features. A sequential implementation of this, for each feature map, would be to scan the input image with a single neuron that has a local receptive field, and to store the states of this neuron at corresponding locations in the feature map[12].

Units in a feature map are constrained to perform the same operation on different parts ot the image. A convolutional layer is usually composed of several feature maps (with different weight vectors), so that multiple features can be extracted at each location.

2.3.2 Pooling/Sub-sampling layer

Once a feature is detected, its' exact position becomes less important as long as its' approximate position relative to other features is preserved. Besides, since the dimensionality of applying a filter is equal to the input dimensionality, we would not be gaining any translation invariance with these additional filters, we would be stuck doing pixel-wise analysis on increasingly abstract features. In order to solve this problem, *subsampling* layer is introduced.

Subsampling, or down-sampling, refers to reducing the overall size of a signal. In many cases, such as audio compression for music files, subsampling is done simply for the size reduction [20]. But in the domain of 2D filter outputs, subsampling also can be thought of reducing the sensitivity of the output to shifts and distortions. One of the most applied subsampling methods used in[11], is known as 'max pooling'. This involves splitting up the matrix of filter outputs into small non-overlapping grids (the larger the grid, the greater the signal reduction), and taking the maximum value in each grid as the value in the reduced matrix. By applying such a max pooling layer in between convolutional layers, we can increase spatial abstractness as we increase feature abstractness[20].

2.3.3 Activation functions

To go from one layer to the next, a set of units compute a weighted sum of their inputs from the previous layer and pass the result through a non-linear activation function[9]. There are many possible choices for the non-linear activation functions in a multi-layered network, and the choice of activation functions for the hidden units may often be different from that for the output units. This is because hidden and output units perform different roles[2].

At present, the most popular non-linear function is the Rectified Linear Units (ReLU), which is simply the half-wave rectifier $f(z) = max(z, 0)$. In the past decades, neural nets used smoother non-linearities, such as $tanh(z)$ or $1/(1 + exp(-z))$, but ReLU typically learns much faster in networks with many layers, allowing training of a deep supervised network without unsupervised pre-training[9].

The rectifier activation function allows a network to easily obtain sparse representations. For example, after uniform initialization of the weights, around 50% of hidden units continuous output values are real zeros, and this fraction can easily increase with sparsity-including regularization. Apart from being more biologically plausible, sparsity also leads to mathematical advantages. On the other hand, one may hypothesize that the hard saturation at 0 may hurt optimization by blocking gradient back-propagation. However, conversely, experimental results done byGlorot et al. suggest that hard zeros can actually help supervised training[4].

2.3.4 Local Response Normalization

ReLUs have the desirable property that they do not require input normalization to prevent them from saturating. If at least some training examples produce a positive input to a ReLU, learning will happen in that neuron. However, we still find that the following local normalization(LRN) scheme aids generalization. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels[8].

This scheme bears some resemblance to the local contrast normalization scheme proposed by Jarrett et al. in [7] without mean activity subtraction which has led to error rate reduction in [8] and [6].

2.3.5 Fully connected/Inner product layer

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via *fully connected layers*(IP). A fully connected layer takes all neurons in the previous layer (be it fully connected, pooling, or convolutional) and connects it to every single neuron it has. Fully connected layers are not spatially located anymore (you can visualize them as one-dimensional), so there can be no convolutional layers after a fully connected layer.

2.4 Model Optimization

. Here, we describe in short some optimization methods along with definition of hyper-parameters used in our model.

2.4.1 Stochastic Gradient Descent

It has often been proposed to minimize the *empirical risk*(training set performance measure. For more detailed description, see [21]) using *gradient descent*(GD)[3]. The standard gradient descent algorithm updates the parameters θ of the objective *J(θ)* as,

$$\theta = \theta - \alpha \bigtriangledown_{\theta} E[(J(\theta)] \tag{2.1}$$

where the expectation in the above equation is approximated by evaluating the cost and gradient over the full training set(Empirical Risk Minimization(ERM)). Stochastic Gradient Descent(SGD) simply does away the expectation in the update and computes the gradient of the parameters using only a single or a few training examples. The new update is given by,

$$\theta = \theta - \alpha \bigtriangledown_{\theta} J(\theta; x^{(i)}, y^{(i)}) \tag{2.2}$$

with a pair $(x^{(i)}, y^{(i)})$ from the training set[15].

Generally each parameter update in SGD is computed with respect to a few training examples or a mini-batch as opposed to a single example. The reason for this is twofold[15]:

1. This reduces the variance in the parameter update and can lead to more stable convergence.

2. This allows the computation to take advantage of highly optimized matrix operations that should be used in a well vectorized computation of the cost and gradient. A typical mini-batch size is 256, although the optimal size of the mini-batch can vary for different applications and architectures.

3. One final but important point regarding SGD is the order in which we present the data to the algorithm. If the data is given in some meaningful order, this can bias the gradient and lead to poor convergence. Generally a good method to avoid this is to randomly shuffle the data prior to each epoch of training.

2.4.2 Weight Decay

As a part of BP algorithm and a subset of regularization methods, *weight decay* adds a penalty term to the error function by multiplying weights to a factor slightly less than 1 after each update.

It has been observed in numerical simulations that a weight decay can improve generalization in a feed-forward neural network. It is proven that a weight decay has two effects in a linear network. First, it suppresses any irrelevant components of the weight vector by choosing the smallest vector that solves the learning problem. Second, if the size is chosen right, a weight decay can suppress some of the effects of static noise on the targets, which improves generalization quite a lot[14].

2.4.3 Momentum

The *momentum* method introduced by [Polyak, 1964] is a first-order optimization method for accelerating gradient descent that accumulates a velocity vector in directions of persistent reduction in the objective across iterations. Given an objective function $f(\theta)$ to be minimized, momentum is given by:

$$\nu_{t+1} = \mu\nu_t - \varepsilon\bigtriangledown \tag{2.3}$$

$$\theta_{t+1} = \theta_t + \nu_{t+1} \tag{2.4}$$

where $\varepsilon > 0$ is the learning rate, $\mu \in [0, 1]$ is the momentum coefficient, and $\bigtriangledown f(\theta_t)$ is the gradient at θ_t [18].

For example, if the objective has form of a long shallow ravine leading to the optimum and steep walls on the sides, standard SGD will tend to oscillate across the narrow ravine since the negative gradient will point down one of the steep sides rather than along the ravine towards the optimum. The objectives of deep architectures have this form near local optima and thus standard SGD can lead to very slow convergence particularly after the initial steep gains. Momentum is one method for pushing the objective more quickly along the shallow ravine[15].

References

[1] Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.

[2] Bishop, C. M. (1995). *Neural networks for pattern recognition.* Oxford university press.

[3] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

[4] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323.

[5] Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. Book in preparation for MIT Press.

[6] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580.*

[7] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE.

[8] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[9] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[10] LeCun, Y. et al. (1989). Generalization and network design strategies. *Connections in Perspective. North-Holland, Amsterdam*, pages 143–55.

[11] LeCun, Y., Jackel, L., Bottou, L., Brunot, A., Cortes, C., et al. (1995). Comparison of learning algorithms for handwritten digit recognition. In *Comparison of learning algorithms for handwritten digit recognition.*

[12] LeCun, Y., Kavukcuoglu, K., Farabet, C., et al. (2010). Convolutional networks and applications in vision. In *ISCAS*, pages 253–256.

[13] Lempitsky, V. and Zisserman, A. (2010). learn. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 1324–1332. Curran Associates, Inc.

[14] Moody, J., Hanson, S., Krogh, A., and Hertz, J. A. (1995). A simple weight decay can improve generalization. *Advances in neural information processing systems*, 4:950–957.

[15] Ng, A. and colleagues (2013). http://ufldl.stanford.edu/tutorial/supervised/ optimization-stochasticgradientdescent/.

[16] Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17.

[17] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

[18] Sutskever, I., Martens, J., Dahl, G., and Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1139–1147.

[19] Tutorial, D. L. (2014). Lisa lab. *University of Montreal.*

[20] University, S. (2013). http://white.stanford.edu/teach/index.php/an_introduction_to _convolutional_neural_networks.

[21] Vapnik, V. N. and Vapnik, V. (1998). *Statistical learning theory*, volume 1. Wiley New York.

[22] Williams, D. R. G. H. R. and Hinton, G. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.