Neural Networks 42 (2013) 83–93

Contents lists available at SciVerse ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Learning in compressed space

Alexander Fabisch [a,*], Yohannes Kassahun [a], Hendrik Wöhrle [b], Frank Kirchner [a,b]

[a] *University of Bremen, Fachbereich 3 - Mathematik und Informatik, Postfach 330 440, 28334 Bremen, Germany*

[b] *Robotics Innovation Center, German Research Center for Artificial Intelligence (DFKI), Robert-Hooke-Str. 5, 28359 Bremen, Germany*

ARTICLE INFO

Article history:
Received 1 October 2012
Revised and accepted 29 January 2013

Keywords:
Model compression
Compressed sensing
Artificial neural networks
Supervised learning
Reinforcement learning

ABSTRACT

We examine two methods which are used to deal with complex machine learning problems: compressed sensing and model compression. We discuss both methods in the context of feed-forward artificial neural networks and develop the backpropagation method in compressed parameter space. We further show that compressing the weights of a layer of a multilayer perceptron is equivalent to compressing the input of the layer. Based on this theoretical framework, we will use orthogonal functions and especially random projections for compression and perform experiments in supervised and reinforcement learning to demonstrate that the presented methods reduce training time significantly.

1. Introduction

Artificial intelligence is facing real world problems and thus machine learning problems become more and more complex. Bengio and Lecun (2007) state that "a long-term goal of machine learning research is to produce methods that will enable artificially intelligent agents capable of learning complex behaviors with minimal human intervention and prior knowledge". As a result, even more effort is shifted from human to machine.

Complex machine learning problems comprise learning complex behaviors like visual and auditory perception and natural language processing as mentioned by Bengio and Lecun (2007) as well as dealing with highly noisy data such as electroencephalography (EEG) signals and learning behavior and perception for complex robots with many actuators and sensors. Examples for complex robots that have many actuators and sensors are ASGUARD (Eich, Grimminger, & Kirchner, 2008), SCORPION (Spenneberg & Kirchner, 2007) and SpaceClimber (Bartsch et al., 2010; Römmerman, Kühn, & Kirchner, 2009). A common characteristic of complex problems is an associated large amount of data, which includes a large input space dimension and/or a large training set. For example, vision problems usually have hundreds to millions of input components as well as thousands of training examples to cover all possible distortions and a typical brain–computer interface (BCI) (Wolpaw, Birbaumer, McFarland, Pfurtscheller, & Vaughan, 2002) can sample 128 channels with 5 kHz.

Complex problems usually result in long training times. We will now consider ways that reduce the training time. There are numerous ways of dealing with large input spaces that reduce the training time. Examples include methods of feature selection (Guyon & Elisseeff, 2003), dimensionality reduction or feature extraction. These methods require in most cases at least some domain specific knowledge and a manually designed preprocessing flow. But developments that reduce the need for human expertise and intervention such as convolutional neural networks (CNNs) (LeCun & Bengio, 1995) or deep belief neural networks (DBNs) (Bengio, 2007) exist as well. The first layers of these kinds of neural networks can be seen as trainable feature extractors, which can be used to cope with large input spaces. This way of dealing with machine learning problems is better because information cannot be lost unintentionally when (almost) unprocessed data is used as input for the learning algorithm.

Another way of avoiding unintentional loss of information is using conventional machine learning algorithms without built-in preprocessing on raw data. But this is computationally more challenging and requires lots of optimizations, in particular during the training. In this article we want to present novel approaches that can deal with complex machine learning problems. In particular, we want to show how to reduce the time needed to train feed-forward neural networks.

We consider only multilayer perceptrons (MLPs) here to demonstrate two ways of simplifying machine learning optimization problems: model compression and data compression. Advantages of MLPs are that they are universal function approximators (Hornik, Stinchcombe, & White, 1989) and they are simple in concept, well-known and widely used. Since MLP layers are contained

* Corresponding author. Tel.: +49 1633143315.
E-mail addresses: afabisch@googlemail.com, afabisch@informatik.uni-bremen.de (A. Fabisch), kassahun@informatik.uni-bremen.de (Y. Kassahun), Hendrik.Woehrle@dfki.de (H. Wöhrle), Frank.Kirchner@dfki.de (F. Kirchner).

0893-6080/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2013.01.020

in CNNs and the backpropagation method is also used in DBNs, we think that this restriction is justifiable. In addition, the presented concepts can be extended to many other learning algorithms. One of these learning algorithms is the support vector machine (SVM) (Boser, Guyon, & Vapnik, 1992; Vapnik, 1995). We can combine SVMs and data compression easily and use SVMs to compare them to MLPs in this article.

The source code for most of the experiments we present here is available online at https://github.com/AlexanderFabisch/OpenANN.

2. Related work

We will first give a short overview of the foundations of our work, which include two research branches. In addition, we will discuss related ideas.

We call the first research branch "model compression". Schmidhuber (1995, 1997) developed a universal *network encoding language* (NEL) to represent neural networks in a compressed form for supervised learning. This was motivated by the search for the best generalizing network, because the simplest hypothesis that fits the training instances should give the best generalization for the latent function. Since the NEL is not continuous it was not possible to use efficient optimization algorithms for this compressed representation. This was the reason for Koutník, Gomez, and Schmidhuber (2010a, 2010b) to develop a continuous representation, where the weights of a recurrent neural network with fixed topology are represented by coefficients of an inverse discrete cosine transform. The focus of their work was reinforcement learning (Sutton & Barto, 1998). Therefore, they presented experiments with the evolutionary optimization algorithm CoSyNE (Gomez, Schmidhuber, & Miikkulainen, 2008). The main goal in this research is the increase of sample efficiency, that is the reduction of episodes needed to learn a good or successful policy. We will present a very similar approach in this article. However, there will be some differences:

- We will focus on feed-forward neural networks, and hence extend the standard backpropagation procedure for learning in compressed space.
- Not only a combination of orthogonal cosine functions, but any kind of orthogonal functions and even randomly generated values can be used to generate the weights. Hence, we will provide a more general framework for model compression.
- Koutník et al. (2010a, 2010b) compress all weights of a neural network with the same coefficients, although they already mention that it would be better to have a distinct set of coefficients for each neuron. We will use a distinct set of coefficients for each neuron to generate all incoming weights.

The second research branch is the so-called "compressed sensing" (Candès & Romberg, 2005; Donoho, 2006). In compressed sensing, sparse or compressible data is examined. It is possible to compress compressible data through *random projections* and reconstruct it with high probability by solving an optimization problem. In combination with machine learning it can be used as a preprocessing method that requires almost no prior knowledge. In particular, the method does not need to compute any features from the training set. Compressed sensing has for example been combined with support vector machines by Calderbank, Jafarpour, and Schapire (2009) and least squares regression by Maillard and Munos (2009). In these cases the reconstruction is not necessary. It is instead assumed that it is possible to distinguish the instances in compressed space because we could reconstruct the original data with high probability. This is true because random projections approximately preserve distances between instances Bingham and Mannila (2001). Random projections are a very powerful technique to overcome the curse of dimensionality when approximate solutions are sufficient. This has for example been proven for similarity search as well (see locality-sensitive hashing (Indyk & Motwani, 1998, Gionis, Indyk, & Motwani, 1999)).

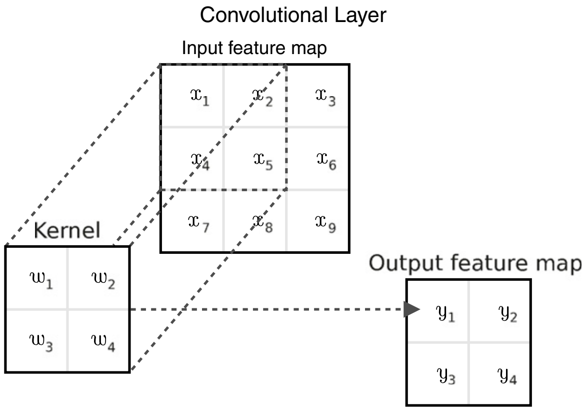

Fig. 1. Convolutional layer of a convolutional neural network. The input feature map is convolved with a parameterizable kernel to create the output. For example, the first entry of the output feature map is $y_1 = w_1x_1 + w_2x_2 + w_3x_4 + w_4x_5$.

We have already shown in a previous paper (Kassahun, Wöhrle, Fabisch, & Tabie, 2012) that compressed sensing and compressing the weights of a neural network that has no hidden layer (single layer perceptron, SLP) are mathematically equivalent. As a contribution to the state of the art, we will show that compressing the weight matrix of any fully connected layer is equivalent to compressing the input to that layer.

Our goal is to reduce training time by reducing the number of parameters that we have to optimize for a neural network. There are other approaches in machine learning that have the same goals and could be combined with the methods we present here. Two ways to do this with neural networks are weight sharing and sparse connections, which are actually forms of model compression. Prominent examples that combine both methods are CNNs (LeCun & Bengio, 1995). These have been particularly successful in recognizing objects in images such as traffic signs (Ciresan, Meier, Masci, & Schmidhuber, 2011). All kinds of CNNs consist of at least one convolutional layer. A very simple example is shown in Fig. 1. In this convolutional layer, we have a two-dimensional input with 3×3 entries and a two-dimensional output with 2×2 entries. These are called feature maps. The input is convolved with a 2×2 kernel to obtain the output. We can construct a fully connected layer, that computes the same output through

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \boldsymbol{W} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_9 \end{pmatrix}, \tag{1}$$

where $\boldsymbol{W}$ is the weight matrix

$$\boldsymbol{W} = \begin{pmatrix} w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 & 0 \\ 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 & 0 \\ 0 & 0 & 0 & 0 & w_1 & w_2 & 0 & w_3 & w_4 \end{pmatrix}. \tag{2}$$

It is obvious that $\boldsymbol{W}$ is sparse and the weights $w_1, \ldots, w_4$ occur in more than one row in this matrix, that is, they are shared among neurons. As a result, we can say that the weight matrix $\boldsymbol{W}$ is generated by a transformation of the convolution kernel. We can make a similar statement about any kind of convolutional layer: the weight matrix of a convolutional layer is generated from less parameters by a transformation. Why this is a form of model compression will become clear at the end of Section 4.1.