INCREMENTAL LEARNING OF CONVOLUTIONAL NEURAL NETWORKS

Dušan Medera
Fac. of Electrical Engineering and Informatics, Technical University of Košice, 042 00 Košice, Slovakia
medera@neuron.tuke.sk

Štefan Babinec
Fac. of Chemical and Food Technologies, Slovak University of Technology, 812 37 Bratislava, Slovakia
stefan.babinec@stuba.sk

Keywords: Convolutional Neural Networks, Incremental Learning, Handwritten Numbers Classification

Abstract: Convolutional neural networks provide robust feature extraction with ability to learn complex, high-dimensional non-linear mappings from collection of examples. To accommodate new, previously unseen data, without the need of retraining the whole network architecture we introduce an algorithm for incremental learning. This algorithm was inspired by AdaBoost algorithm. It utilizes ensemble of modified convolutional neural networks as classifiers by generating multiple hypotheses. Furthermore, with this algorithm we can work with the confidence score of classification, which can play crucial importance in specific real world tasks. This approach was tested on handwritten numbers classification. The classification error achieved by this approach was highly comparable with non-incremental learning.

1 INTRODUCTION

Many claims have been made about the importance of neural networks in modeling nature and artificial intelligence problem domains. To this stage, the neural networks have been applied to a variety of areas. One of the most studied areas, where neural networks were successfully applied and where they still have a potential, is pattern recognition task. The performance of neural networks as classifiers relies heavily on the availability of a representative set of the training examples. In many practical applications, data acquisition and the training process is time consuming. It is not uncommon in applications, that the data are available in small batches over a period of time. In ideal case, the classifier should support an incremental fashion of accommodating new data without compromising old data classification performance. Learning new information without forgetting previously acquired knowledge raises so-called stability-plasticity dilemma (Grosberg, 1988). The dilemma points out the fact that a completely stable classifier will preserve existing knowledge, but will not accommodate any new information. On the other hand a completely plastic classifier will learn new information and will not conserve prior knowledge.

Convolutional neural networks (CNN) are representative of classifiers, which require retraining of the classifier using all data that have been accumulated so far. Why there is an effort to accommodate convolutional neural networks to incremental fashion? They have been widely adopted in pattern recognition (Y. LeCun and Haffner, 1998) and image recognition areas (Delakis and Garcia, 2003). From their fundamental principles they lack the option to accommodate new unseen data without time consuming learning process, which can play a crucial role in many applications (e.g. face recognition).

In this paper we introduce incremental learning algorithm of convolutional neural networks, which was inspired by AdaBoost and Learn++ algorithm. This algorithm allows us to accommodate new previously unseen data without the need of retraining the whole network architecture. It utilizes ensemble of modified convolutional neural networks as classifiers by generating multiple hypotheses. Furthermore, with this algorithm we can work with the confidence score of classification, which can play crucial importance in specific real world tasks. Classification results of this incremental approach are better quality as the results gained from the non-incremental approach.

2 BACKGROUND

2.1 Convolutional Neural Networks

The ability of multi-layer neural networks trained with gradient descent to learn complex, high-dimensional, non-linear mappings from collection of examples makes them candidates for image recognition tasks. In the traditional model of pattern recognition, a hand-designed feature extractor gathers relevant information from the input and eliminates irrelevant variabilities. A trainable classifier then categorizes the resulting feature vectors into classes. In this scheme standard fully connected multi-layer networks can be used as classifiers. A potentially more interesting scheme is relating on learning in the feature extractor itself as much as possible.

CNN combines three architectural ideas to ensure some degree of shift, scale and distortion invariance: local receptive fields, shared weights and temporal sub-sampling. In the Figure 1 we can see the concrete CNN architecture used in our experiments (inspired by Yann LeCun (Y. LeCun and Haffner, 1998)). The CNN is combined from pairs of convolutional (CL) and sub-sampling (SL) layers followed by hidden layer with full connectivity to the layer of distributed codes. Each unit in CL or SL layer receives inputs from a set of units located in a small neighborhood in the previous layer. The idea of using local receptive fields allows extraction of elementary visual features in the CL layers. These features are then combined by SL layers in order to detect higher-order features. Units in a layer are organized in planes within which all the units share the same set of weights (receptive field). The set of outputs of the units in a plane is called a feature map. Feature map units are all constrained to perform the same operation on different parts of the image. A CL layer is composed of feature maps, so they can extract multiple features. Sub-sampling layers SL perform a local averaging and reducing the resolution of the feature map, so they are reducing the sensitivity of the input to the shifts and distortions. A large degree of invariance to geometric transformations of the input images can be achieved with reduction of spatial resolution compensated by increasing the number of feature maps. Finally, computing the gradient of the loss function with respect to all the weights in CNN is done with slightly modified back-propagation algorithm.

Output layer consists of Euclidean radial basis function (RBF) neurons. The reason for using RBF neurons is to connect distributed codes layer with classification classes in output layer. Final outputs of neurons in output layer are calculated from the formula

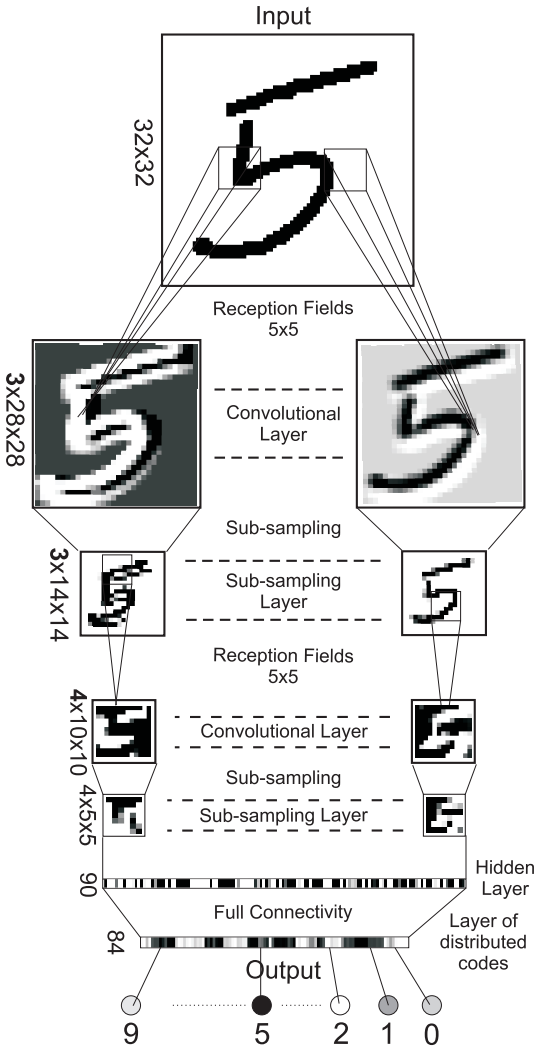

Figure 1: Convolutional neural network used in our experiments.

$$y_i = \sum_j (x_j - w_{ij})^2, \tag{1}$$

where y_i is the output of ith neuron in the output layer, x_j is the output of jth neuron in the distributed codes layer and w_{ij} is the synaptic weight between neurons in these two layers. In other words, we are computing Euclidean distance between the input vector and the synaptic weights vector.

Classification confidence score is very important in the decision making support systems. We can determine it in the following way. Neurons in the output layer can be considered as centers of individual classes of clusters defined through values of the synaptic weights. We can use the following Gaussian

function for our purpose (Duin and Tax, 1998):

$$\varphi(y_i) = e^{-\frac{y_i}{\sigma}}, \quad (2)$$

where σ is the radius of the cluster defined through distributed codes. It is clear that values of the $\varphi(y_i)$ function are from the interval $[0,1]$ and we can consider these values as a confidence score. For the output, which is close to the distributed code of the corresponding class, confidence will be close to 1 and vice versa.

2.2 Incremental Learning

We should note that the term "incremental learning" has been used rather loosely in the literature, where the term also referred to incremental network growing and pruning or on-line learning. Furthermore, various other terms, such as constructive learning, lifelong learning and evolutionary learning have also been used to imply learning new information.

Pure incremental learning is important as a model of how learning occurs in real, biological brains. Humans and other animals are often observed to learn new things without forgetting old things. But pure sequential learning tends not to work well in artificial neural networks. Using a fixed architecture, distributed representation and a training algorithm based on minimizing an objective function results sequential learning in "catastrophic interference". The reason is that the minima of the objective function for one training set may be totally different than the minima for subsequent training sets. Hence each successive training set may cause the network to forget completely all previous training sets. This problem is also called the "stability-plasticity dilemma" mentioned above.

We can define an incremental learning algorithm as one that meets the following criteria (R. Polikar and Udpa, 2001):

1. It should be able to learn additional information from new data.
2. It should not require access to the original data used to train the existing classifier.
3. It should preserve previously acquired knowledge.
4. It should not suffer from catastrophic forgetting.
5. It should be able to accommodate new classes that may be introduced with new data.

An algorithm that covers all properties would be a universal tool for pattern recognition and machine learning. Many applications can benefit from versatile incremental learning algorithm.

2.3 Boosting

Boosting refers to a general and provably effective method of producing a very accurate prediction rule by combining rough and moderately inaccurate rules of thumb in a manner similar to that suggested above. Boosting has its roots in a theoretical framework for studying machine learning called the "Probably Approximately Correct (PAC)" learning model due to Valiant (Valiant, 1984). Valiant was first to pose the question of whatever a "weak" learning algorithm which performs just slightly better that random guessing in the PAC model can be boosted into an accurate "strong" learning algorithm. We focus on the AdaBoost algorithm introduced by Freund and Schapire in 1995 (Freund and Schapire, 1999).

The algorithm takes as input a training set $(x_1;y_1),\ldots,(x_m;y_m)$, where each x_i belongs to some domain or instance space X, and each label y_i is in some label set Y. The basic version of AdaBoost algorithm preserves $Y = \{-1,+1\}$. There were made enhancements to enable multiclass classification. AdaBoost calls a given weak or base learning algorithm repeatedly in a series of rounds $t = 1,\ldots,T$. One of the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. The weight of this distribution on training example i on round t is denoted $D_t(i)$. Initially, all weights are set equally, but on each round are increased the weights of incorrectly classified examples so that the weak learner is forced to focus on the hard examples in the training set. The weak learner's job is to find a weak hypothesis $h_t : X \to \{-1,+1\}$ appropriate for the distribution D_t. The goodness of a weak hypothesis is measured by its error, which is measured with respect to the distribution D_t on which the weak learner was trained. In practice, the weak learner may be an algorithm that can use the weights D_t on the training examples. This is an important note that "weak" learner can be replaced by neural network. Once the weak hypothesis h_t has been received, AdaBoost chooses a parameter α_t that measures the importance of the weak hypotheses h_t. The final hypothesis H is a weighted majority vote of the T weak hypotheses where α_t is the weight assigned to h_t.

Practically, AdaBoost has many advantages. It requires no prior knowledge about the weak learner and so can be flexibly combined with any method for finding weak hypotheses. Finally, it comes with a set of theoretical guarantees given sufficient data and a weak learner that can reliably provide only moderately accurate weak hypotheses. On the other hand, the actual performance of boosting on a particular problem is clearly dependent on the data and

the weak learner. Consistent with theory, boosting can fail to perform well given insufficient data, overly complex weak hypotheses or weak hypotheses which are too weak. Boosting seems to be especially susceptible to noise. Boosting approach was also used to improve classification performance on convolutional neural networks (Y. LeCun and Haffner, 1998).

3 SYSTEM ARCHITECTURE

3.1 Ensemble of Classifiers

To follow an approach to the incremental learning problem, while using supervised feed-forward networks, we are facing a problem, how to implement in the process criteria mentioned above. One possible solution is to generate multiple new classifiers for previously unseen portions of the feature space instead of generating new cluster nodes for each previously unseen instance.

The proposed incremental learning system using Convolutional Neural Networks described in this section was inspired by the AdaBoost algorithm (Freund and Schapire, 1999) and Learn++ algorithm (R. Polikar and Udpa, 2001). Learn++ algorithm was designed for incremental learning of supervised neural networks, such as multilayer perceptrons to accommodate new data without access to previously trained data in the learning phase. Algorithm generates an ensemble of weak classifiers, each trained using a different distribution of training samples. Outputs of these classifiers are then combined using Littlestone's majority-voting scheme to obtain the final classification rule.

Ensemble of classifiers can be optimized for improving classifier accuracy or for incremental learning or new data (Polikar, 2007). Combining ensemble of classifiers is geared towards achieving incremental learning besides improving the overall classification performance according to the boosting. Proposed architecture is based on this intuition: each new classifier added to the ensemble is trained using a set of examples drawn according to a distribution, which ensures that examples that are misclassified by the current ensemble have a high probability of being sampled (examples with high error rates are precisely those that are unknown).

3.2 Using Convolutional Neural Network as a Weak Learner

Neural networks can simulate a weak learner, when their architecture is kept undersized or their error goal is kept high with respect to the complexity of the particular problem. In average, weak learners should perform over 50% and their performance is improved (boosted) when hypotheses are combined together. Convolutional neural networks were developed in more sophisticated structures, which contain multiple hidden layers organized into planes, by origin. We can simulate such a kind of neural network as a weak learner in two approaches. First we consider structural changes that will lead to the simplifying of the overall structure, such as reducing number of planes in convolutional layers and sub-sampling layers. Second, the network can by trained with a larger error goal. Different architectures and error goals will reflect in algorithm sensitivity and invariance, learning capabilities and classification results in the incremental manner.

3.3 Incremental Learning

We can describe the learning algorithm inspired by (R. Polikar and Udpa, 2001) by assuming following inputs. Denote training data $S_k = [(x_1, y_1), \ldots, (x_m, y_m)]$, where x_i are training samples and y_i are corresponding correct labels for m samples randomly selected from the database Ω_k, where $k = 1, 2, \ldots, K$. Algorithm calls weak learner repeatedly to generate multiple hypotheses using different subsets of the training data S_k. Each hypothesis learns only a portion of input space X and is weighted according to the final hypothesis. The weight of distribution on training example i on round t is denoted $D_t(i)$. Those weights are initialized by rule $D_1(i) = w_1(i) = \frac{1}{m}$, unless there is a prior knowledge to select otherwise. Learning process is iterative and in each iteration $t = 1, 2, \ldots, T_k$, where T_k is number of classifiers (weak learners used in current iteration), algorithm dichotomizes S_k into a training subset TR_t and test subset TE_t according to D_t. All classifiers are called and the hypothesis $h_t : X \rightarrow Y$ is generated. The error of h_t on S_k is defined as

$$\varepsilon_t = \sum_{i:h_t(x_i) \neq y_i} D_t(i), \tag{3}$$

which is simply the sum of distribution weights of misclassified examples. If $\varepsilon_t > \frac{1}{2}$, h_t is discarded and we repeat this step. Otherwise we compute normalized error β_t as

$$\beta_t = \frac{\varepsilon_t}{1 - \varepsilon_t}. \tag{4}$$

All hypotheses generated in the previous t iterations are then combined using weighted majority voting scheme to obtain composite hypothesis (hypothesis performs well on own training and testing data set by

giving them larger voting powers)

$$H_t = arg\max_{y\in Y} \sum_{t:h_t(x)=y} log\frac{1}{\beta_t}. \quad (5)$$

The composite error made by hypothesis H_t is computed by following equation:

$$E_t = \sum_{t:H_t(x_i\neq y_i)} D_t(i). \quad (6)$$

If $E_t > \frac{1}{2}$, current h_t is discarded and a new TR_t and TE_t is selected to obtain new h_t. E_t can only exceed this threshold during the iteration after new database Ω_{k+1} is introduced. If $E_t < \frac{1}{2}$, composite normalized error is computed as

$$B_t = \frac{E_t}{1-E_t}. \quad (7)$$

After computing B_t, the weights $w_t(i)$ are adjusted in an incremental manner of the algorithm

$$\begin{array}{ll} w_{t+1}(i) = w_t(i)B_t & \text{if} \quad H_t(x_i) = y_i, \\ w_{t+1}(i) = w_t(i) & \text{otherwise.} \end{array} \quad (8)$$

In other words, if an example x_i is correctly classified by H_t, its weight is multiplied by B_t, otherwise the weight is kept unchanged. This rule reduces the probability of correctly classified examples being chosen to TR_{t+1}. Using of composite hypothesis makes incremental learning possible particularly in cases when examples from the new class are introduced.

Finally after T_k hypotheses are generated for each Ω_k, the final hypothesis is obtained by the weighted majority voting of all composite hypotheses:

$$H_{final} = arg\max_{y\in Y} \sum_{k=1}^{K} \sum_{H_t(x)=y} log\frac{1}{B_t}. \quad (9)$$

Incremental learning is achieved through generating additional classifiers and former knowledge is not lost since all classifiers are retained.

4 Experiments

Our goal in this paper was to compare results achieved by non-incremental learning of CNN with our new approach. We have used standard benchmarking data set MNIST in this paper. This data set represents handwritten numbers samples from different people. Every sample from this data set is represented in grayscale with 28×28 dimension and is centered in the 32×32 grid.

The configuration of CNN's architecture used in our experiments is shown in the following Table 1.

Table 1: Architecture of the CNN used in our experiments.

Dim. of input layer	Reception fields	Number of maps	Dim. of conv. layer
32x32	5x5	3	28x28
		4	10x10
Dimension of sub-sampling layer	**Hidden layers**	**Distrib. codes layer**	**Output layer**
14x14	1x90	84	10
5x5			

We have created 4 independent training sets D1, D2, D3 and D4. Every training set was composed of 2500 samples and the testing set was composed of the next 5000 samples.

We have used modified Levenberg-Marquadt backpropagation of error method (LeCun, 1998) as a learning algorithm for the individual classifiers. Regarding to smaller training set, the number of learning cycles was set to 20. The initial value of the global learning parameter γ in learning algorithm equals 5.10^{-5}. In the 4th cycle the value was decreased to 2.10^{-5} and in the 12th cycle to 1.10^{-5}. The value of the parameter μ was set to 0.02. Maximum number of classifiers per data set was constrained to 5 for every training data set.

To allow comparison with non-incremental approach we have also trained one CNN, which was still retrained for all gradually presented training data sets. We have retrained it 4 times, where one retraining consisted of 40 learning cycles.

We can see the results of experiments in the folowing Tables 2 and 3.

Best results were achieved with the cluster radius parameter σ set to 1. As we can see, our incremental approach achieved better results during the whole gradual training process.

5 Conclusions

Incremental learning inspired by Learn++ algorithm is based on ensemble of Convolutional Neural Network classifiers. Algorithm's update rule is optimized for incremental learning of new data. It does not require access to previously seen data during subsequent training process and it is able to retain previously acquired knowledge.

Table 2: Results of experiments for standard non-incremental learning.

Non-incremental approach		
Prediction accuracy		
	Training set [%]	Testing set [%]
D1	92.13	88.07
D1 D2	91.42	87.76
D1 D2 D3	91.99	89.36
D1 D2 D3 D4	92.12	89.63

Table 3: Results of experiments for our new approach.

Our new approach			
Number of classifiers	Prediction accuracy		
		Training set [%]	Testing set [%]
1	D1	95.52	91.95
2		98.88	92.76
3		99.44	93.02
4		99.56	92.87
5		99.56	92.76
6	D2	95.36	93.80
7		98.04	94.14
8		99.32	94.12
9		99.40	94.00
10		99.56	93.98
11	D3	98.04	94.70
12		99.60	94.90
13		99.76	94.92
14		99.84	94.86
15		99.84	94.82
16	D4	97.32	95.18
17		99.20	95.42
18		99.68	95.62
19		99.96	95.62
20		99.96	95.59

There are still some issues for further research. For example proper choice of algorithm parameters, such as base convolutional neural networks classifiers, error goal or number of generated hypotheses.

We have chosen handwritten numbers set as a testing data for our classification experiments. Our aim was to find out if this approach can give better results in comparison with standard non-incremental algorithm and in the same time offer incremental form of learning. From the results shown in this paper, it is clear that this aim has been accomplished. Classification results of our incremental algorithm are better as the results gained from the non-incremental approach. In addition, our approach brings the possibility to work with the confidence score of classification. Incremental learning inspired by Learn++ algorithm is based on ensemble of Convolutional Neural Network classifiers. Algorithm's update rule is optimized for incremental learning of new data. It does not require access to previously seen data during subsequent training process and it is able to retain previously acquired knowledge.

ACKNOWLEDGEMENTS

This work was supported by Scientific Grant Agency Vega of Slovak Republic under grants 1/4053/07, 1/0804/08 and 1/0848/08.

REFERENCES

Delakis, M. and Garcia, C. (2003). Training convolutional filters for robust face detection. In *Proc. of the IEEE international Workshop of Neural Networks for Signal Processing*, pages 739–748.

Duin, R. and Tax, D. (1998). Efficient backprop, neural networks tricks of the trade. *Classifier Conditional Posterior Probabilities*, 1451:611–619.

Freund, Y. and Schapire, R. (1999). A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14:771–780.

Grosberg, S. (1988). Nonlinear neural networks principles, mechanisms and architectures. *Neural Networks*, 1(1):17–61.

LeCun, Y. (1998). Efficient backprop, neural networks tricks of the trade. *Lecture Notes in Computer Science*, 1524:9–53.

Polikar, R. (2007). Bootstrap inspired techniques in computational intelligence. *IEEE Signal Processing Magazine*, 24(4):56–72.

R. Polikar, L. U. and Udpa, S. (2001). Learn++, an incremental learning algorithm for supervised neural networks. *IEEE Trans. on Systems*, 31(4):497–508.

Valiant, L. (1984). A theory of the learnable. *Communications of the ACM*, 27:1134–1142.

Y. LeCun, L. Bottou, Y. B. and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. of IEEE*, 86(11):2278–2324.