Character Recognition Using Convolutional Neural Networks

David Bouchain

Seminar *Statistical Learning Theory*
University of Ulm, Germany
Institute for Neural Information Processing
Winter 2006/2007

Abstract

Pattern recognition is one of the traditional uses of neural networks. When trained with gradient-based learning methods, these networks can learn the classification of input data by example. An introduction to classifiers and gradient-based learning is given. It is shown how several perceptrons can be combined and trained gradient-based. Furthermore, an overview of convolutional neural networks, as well as a real-world example, are discussed.

Keywords: classification, gradient-based learning, backpropagation, convolutional neural networks

1 Introduction

In this paper, the use of gradient-based learning on neural networks, and convolutional neural networks in particular, is discussed. Starting with the concept of the most abstract classifier, the pattern recognition problem is specified in Section 2. Applied to this problem, the technique of gradient-based learning is introduced. This numerical optimization procedure is used to drive a global error function of the classifier to a local minimum.

Then follows an overview of the single perceptron, which constitutes the most basic neural classifier. It is shown how to interconnect several perceptrons to obtain a multilayer perceptron, which can then be trained through the use of the backpropagation algorithm. The backpropagation algorithm has become a widely used, de facto standard for neural pattern recognition.

Eventually, the concept of convolutional neural networks is introduced in Section 3. These networks offer improvement over the multilayer perceptron by means of performance, accuracy and some degree of invariance to distortions in the input images. Finally, the perfor-

Email address: david@bouchain.de (David Bouchain).

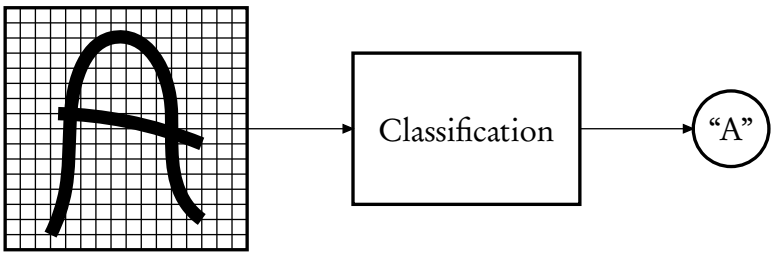

Figure 1. The general classifier. An input image, in this case a hand-written character, is run through the classification function, ideally yielding the correct output value.

mance of convolutional neural networks is analyzed through a real-world example in Section 4, showing that this architecture outperforms most other methods of pattern classification.

2 Gradient-Based Learning

Character recognition, and pattern recognition in general, addresses the problem of classifying input data, represented as vectors, into categories [3]. A general classifier is shown in Figure 1. The main difficulty is to find the optimal or a near-optimal *classification function*. In general, this function can be formalized as

$$\mathbf{y}^p = F(\mathbf{x}^p, W) \tag{1}$$

where $\mathbf{x}^p$ is the p-th input vector (for example, a grayscale image representing a hand-written digit), $\mathbf{y}^p$ is the corresponding output vector, and W is a set of trainable parameters for the classifier [1]. If there exists a vector $\mathbf{T}^p$ representing the “desired” output [1], called the *label* or *teacher signal* [5], for each input vector $\mathbf{x}^p$, then $\mathscr{T}$ is defined as

$$\mathscr{T} = \left\{(\mathbf{x}^p, \mathbf{T}^p) : \mathbf{x}^p \in \mathbb{R}^d, \mathbf{T}^p \in \mathbb{R}^n, p = 1, \ldots, P\right\} \tag{2}$$

and is called the *training set*. Adjusting or *training* the parameters based on the comparison between output $\mathbf{y}^p$ and label $\mathbf{T}^p$ is called *supervised learning* [5].

Determining the amount of adjustment for the parameters W is done through the *loss function* [1] or *error function* [4,5]. It is denoted $E(W)$ for all patterns, or $E^p(W)$ for the p-th pattern [5] and returns the *error* of the classifier. When applied to real-world problems, the training set is often split up into two subsets, one for the training and one to test the actual performance, yielding two error values, e_{test} and e_{train}. The overall goal is then to not only minimize e_{train}, but also to minimize $e_{\text{test}} - e_{\text{train}}$. This so-called *gap* is an indication for how well the classifier performs on input patterns which it has *not* been trained with [1].

One of the most successful machine learning techniques is the class of *gradient-based learning* methods. These methods are widely used especially for pattern recognition [1]. The basic concept is to sequentially adjust the parameters $W = W(t)$, where t is the current training

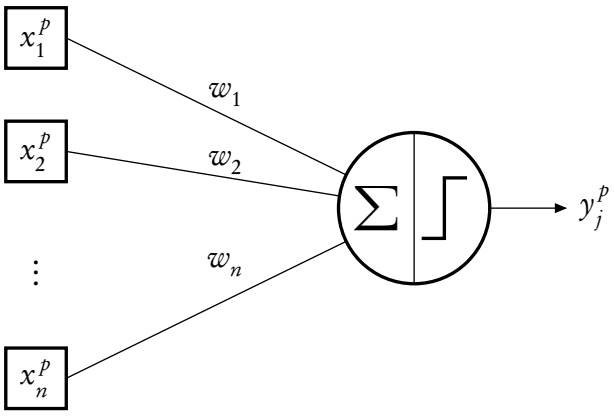

Figure 2. The single perceptron classifies an input vector of dimension n, provided that the given problem is linear-separable.

step, according to the gradient of the scalar-valued error function [5]. By using the rule

$$W(t+1) = W(t) - \varepsilon \frac{\partial E(W(t))}{\partial W(t)} \tag{3}$$

for parameter modification after the presentation of all patterns, or respectively

$$W(t+1) = W(t) - \varepsilon \frac{\partial E^p(W(t))}{\partial W(t)} \tag{4}$$

for adjustment after the presentation of one pattern, the error function will be reduced towards a local minimum. ε is the *learning rate*, which affects how fast a local minimum is found. The following will discuss the application of this general gradient-descent principle for neural networks.

2.1 The Multilayer Perceptron

The single perceptron is a simple computational model of the biological neuron. It was first introduced by Warren McCulloch and Walter Pitts in 1943, and in 1958 refined by Frank Rosenblatt [6]. As shown in Figure 2, the incoming connections from other neurons are represented by an input vector $\mathbf{x}$. All input components are multiplied by their corresponding components in the *weight vector* $\mathbf{w}$ and summed up, yielding the scalar product of $\mathbf{x}$ and $\mathbf{w}$ with *threshold* [6] or *bias* [5] θ:

$$u = \sum_{k=1}^{n} \mathbf{x}_k \mathbf{w}_k - \theta = \langle \mathbf{x}, \mathbf{w} \rangle - \theta \tag{5}$$

The resulting scalar is fed to the *transfer function* [5] or *squashing function* [1] to produce the perceptron's output:

$$y = f(u) \tag{6}$$

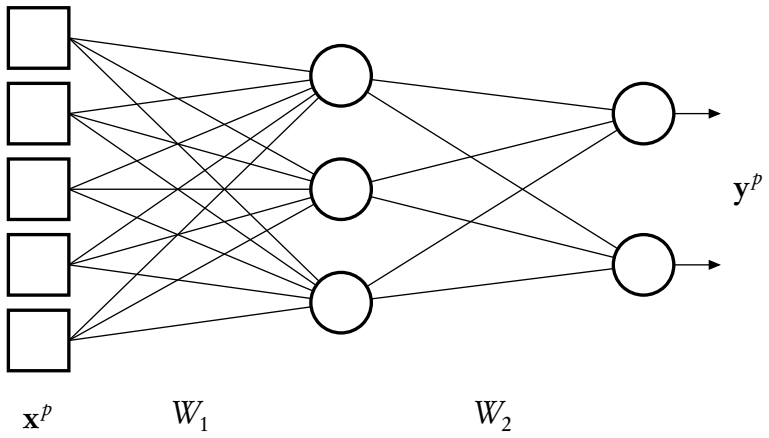

Figure 3. A three-layer neural network. The first layer is the input layer and does not perform any actual computation, but is mapped directly onto the input vector.

For the single perceptron, the transfer function is usually chosen to be the step function or sign function [6,4], with the leap from 0 to 1 (or from -1 to 1, respectively) occuring at the threshold θ. However, for the multilayer perceptron, a sigmoid function like the *Fermi function* or inverse tangent is needed. The reason for this is described below.

This simple perceptron is able to correctly classify (or learn the classification of) all *linear-separable* classes that have its input dimension. In 1960, Rosenblatt introduced the widely-known *perceptron learning algorithm* [6]. The convergence of this algorithm in finite time for linear-separable data was proven by Marvin Minsky and Seymore Papert in 1969 [5].

A set $\mathscr{P}$ of several perceptrons, each of wich classifies the input data differently, can be combined via an additional perceptron which receives all outputs from $\mathscr{P}$ as input. This makes it possible to classify data which is not linear-separable [5]. $\mathscr{P}$ is then called a *neural layer* or simply a *layer* [6]. By chaining two or more layers together, where one layer is fully interconnected with the previous layer, a so-called *multilayer perceptron* or *multilayer neural network* is attained [6]. A simple three-layer perceptron, where the first layer is called the *input layer*, is shown in Figure 3. The last layer in the network is called the *output layer*. All other layers are called *hidden layers* [6]. As data is only propagated forward in this network architecture, a multilayer perceptron is a *feed-forward network* [5].

2.2 Error Backpropagation

The weights in a multilayer perceptron can be grouped into a matrix W of the form

$$W = \left((w_{ij})_{i=1}^{n}\right)_{j=1}^{n} = \begin{pmatrix} 0 & W_1 & 0 \\ 0 & 0 & W_2 \\ 0 & 0 & 0 \end{pmatrix} \tag{7}$$

for a three-layer network, where W_1 and W_2 are the weight matrices between the first and second layer, and between the second and third layer, respectively. w_{ij} thereby denotes the

connection from the i-th to the j-th neuron. Assume that the error function is given as

$$E(W) = \sum_p E^p(W) = \sum_p ||\mathbf{T}^p - \mathbf{y}^p||_2^2 \tag{8}$$

where $\mathbf{y}^p$ is the vector containing the output from the last layer. To lower the error for a specific pattern $\mathbf{x}^{\tilde{p}}$, the gradient of E^p can be used to adjust the weights accordingly. By differentiating E^p,

$$\frac{\partial E^p}{\partial w_{ij}} = -2(t_j - y_j) f'(u_j) y_i \tag{9}$$

is obtained, where T_j and y_j are the respective components of the teacher signal and the output vector. u_j is the *dendrite potential* of neuron j, as defined in Equation 5. The partial derivative shows the direction of increase, so it must be subtracted from the weight w_{ij}. This yields the learning rule for neurons in the output layer

$$w_{ij}(t+1) = w_{ij}(t) + \varepsilon y_i \delta_j \tag{10}$$

where $\delta_j = (T_j - y_j) f'(u_j)$. y_j is the output of neuron j in the output layer, and y_i is the output of neuron i in the previous layer. This shows that the squashing function has to be differentiable. So instead of the sign function or step function, a sigmoid function like

$$f(x) = \frac{1}{1 + e^{-\beta x}} \tag{11}$$

is used. This function is not only continuously differentiable, but it also guarantees that its value is bounded between 0 and 1 (hence the name "squashing" function). The learning rule for the hidden layer is

$$w_{ki}(t+1) = w_{ki}(t) + \varepsilon x_k \delta_i \tag{12}$$

with $\delta_i = \sum_j \delta_j w_{ij} f'(u_i)$. The weight between input layer and hidden layer is deonted by w_{ki}. So, each weight adjustment for the hidden layer takes into account the weights of the output layer. The weights for neuron i in the hidden layer are said to take *responsibility* or *blame* [6] for all errors in the output layer, to the amount of its respective weights to the output neurons. In other words, the error is passed back to the hidden layer, hence the name *backpropagation*.

In 1989, George Cybenko showed that a three-layer neural network, a multilayer perceptron with one hidden layer, can approximate all continuous, real-valued functions to any desired degree [5]. Thus, a three-layer neural network can also approximate any continuous *decision boundary* between two classes to any desired accuracy [5]. With two hidden layers, even discontinuous functions can be represented [6]. This shows that the multilayer perceptron is a

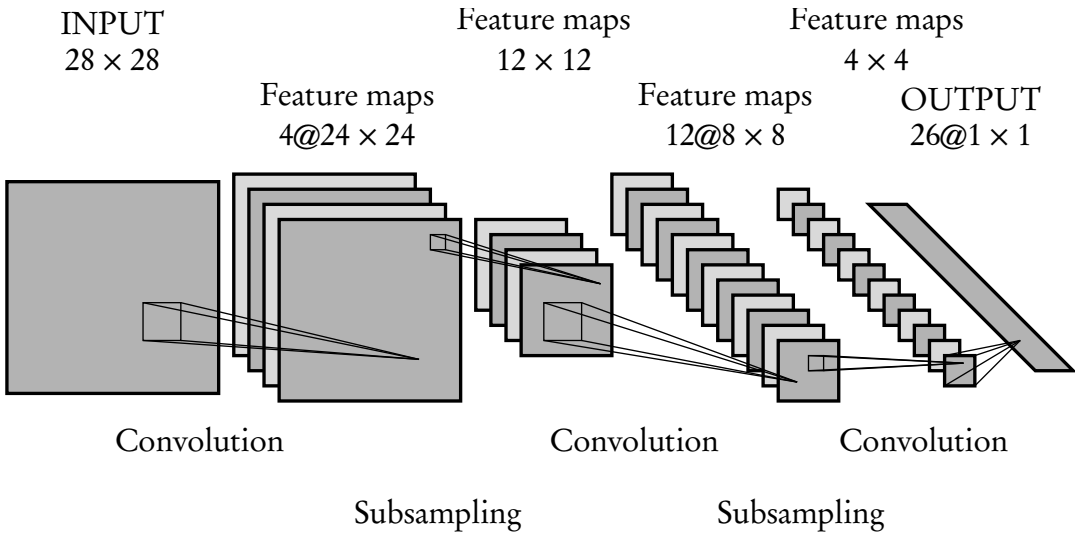

Figure 4. A six-layer convolutional neural network.

powerful tool for learning the classification of high-dimensional data. Of all gradient-based learning methods, the error backpropagation algorithm has emerged as the *de facto* standard for pattern recognition tasks in machine learning [4].

3 Convolutional Neural Networks

While the multilayer perceptron described above performs well for rather simple classification problems, it has several drawbacks when it comes to real-world applications. First, the number of trainable parameters becomes extremely large. For example, a 24×24 input layer would already have 600 connections per single neuron in the hidden layer. Secondly, it offers little or no invariance to shifting, scaling, and other forms of distortion [1]. Third, the topology of the input data is completely ignored, yielding similar training results for all permutations of the input vector. To overcome these problems, a classifier is often split up into a hand-crafted feature extractor and the actual trainable classifier module. Designing the feature extractor by hand requires a lot of heuristics and, most notably, a great deal of time [1].

In 1995, in an attempt to tackle the problems of the multilayer perceptron, Yann LeCun and Yoshua Bengio introduced the concept of *convolutional neural networks* [2], which were neurobiologically motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex of the cat [4]. They designed a network structure that implicitely extracts relevant features, by restricting the neural weights of one layer to a local *receptive field* in the previous layer. Thus, a *feature map* is obtained in the second layer. By reducing the spatial resolution of the feature map, a certain degree of shift and distortion invariance is achieved [4]. Also, the number of free parameters is significantly decreased by using the same set of weights for all features in the feature map [1].

3.1 Topology

A simple convolutional neural network, as first proposed by LeCun and Bengio, is shown in Figure 4. As can be seen, the first convolutional layer contains four feature maps, where each

neuron has a receptive field of 5×5 in the input image. To eliminate the need for boundary processing, the feature maps have a slightly smaller resolution than the input [1]. The first subsampling layer consists of four feature maps, which are scaled local averages of the respective feature maps in the previous convolutional layer. Each neuron in the subsampling layer has one trainable coefficient for the local average, and one trainable bias [1]. The next convolutional and subsampling layers operate in the same manner. However, the second convolutional layer has 12 feature maps, resulting in an increase in feature space [4]. Similarily, the subsampling layers reduce the spatial resolution. The last convolutional layer is the output layer and contains only one feature map [2]. To obtain the final result of the classifier, the neurons of the output layer can then be compared to detect a *winner* neuron [1].

Potentially, each feature map receives input from all feature maps in the previous layer [4]. However, in order to prevent all feature maps from learning the same, this is usually not the case. Section 4 contains a table of feature map interconnections for a real-world example.

3.2 Training

As a special form of the multilayer perceptron, convolutional neural networks are trained through backpropagation [1]. Because all neurons in one feature map of a convolutional layer share the same weights and bias, the number of parameters is dramatically smaller then in a fully interconnected multilayer perceptron, leading to an implicit reduction of the gap $e_{\text{test}} - e_{\text{train}}$. The subsampling layers have one trainable weight (local average coefficient) and one trainable bias [1], so the number of free parameters in the subsampling layers is even lower than in the convolutional layers.

Because for this low number of free parameters, the training of convolutional neural networks requires far less computational effort than the training of multilayer perceptrons. This, as well as the implicit feature extraction and distortion invariance (to some degree), make convolutional networks an obvious candidate for classification tasks, especially pattern recognition. They have successfully been used for various pattern recognition tasks, such as handwriting and face recognition [1].

4 Application: LeNet-5

In [1], LeCun et. al. present a convolutional neural network specifically designed for handwritten digit recognition, named *LeNet-5*. They not only show that LeNet-5 delivers better accuracy than other techniques, but also that it does so at high speed. Particularly with respect to noise resistance and distortion invariance, convolutional neural networks are shown to be the architecture of choice for pattern recognition.

4.1 Topology

LeNet-5 consists of eight layers. The input layer size is 32×32, even though the input images used are at most 20×20. The reason for this is, that the relevant features are then guaranteed to be contained in all feature maps and not get lost because they are near the boundary.

Table 1
The interconnection table of the first subsampling layer with the second convolutional layer. Each X marks a connection, on the left are the feature map numbers of the subsampling layer, on the right those of the convolutional layer.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

The first convolutional layer has six feature maps, each of which has a resolution of 28×28, with a receptive field of 5×5. The second layer, or the first subsampling layer, contains six feature maps of size 14×14. The third layer is another convolutional layer and has 16 feature maps with size 10×10, with a receptive field of 5×5. The fourth layer contains 16 feature maps as well, each of which is of size 5×5. The fifth layer is a convolutional layer with 120 feature maps, again with a receptive field of 5×5. Then, as opposed to the convolutional neural network shown in Figure 4, follows a layer with 84 neurons, which is fully interconnected with the previous layer. All neurons up to and including the sixth layer compute their input by calculating the weighted sum and feeding the result to the squashing function

$$f(u) = A \tanh(Su) \tag{13}$$

where A was chosen to be 1.7159, and S determines the slope of the function at the origin. Finally, the last layer contains ten radial basis function neurons. The weights of the output layer were initialized by hand (as opposed to random for all other weights) to map to a bitmap representation of the ASCII character set, in order to overcome difficulties with characters that are similar, and thus give a similar output.

The third layer, a convolutional layer, was chosen to not have full interconnection with respect to the feature map. Table 1 shows which feature map of the second convolutional layer has connections to which feature map in the first subsampling layer. This is done in order to prevent subsequent feature map to extract the same features from previous layers.

4.2 Performance

The network was tested with a database containing more than 50,000 hand-written digits, all normalized and centered in the input image. An error rate of about 0.95% was achieved. When the network was trained with distorted versions of the input images, the error rate even dropped to 0.8%. These results were compared to various other classifiers.

On a multilayer perceptron with one hidden layer, for example, the error rate was about 4.5%. With two hidden layers, a test error rate of 3.05% was calculated with the first hidden layer

consisting of 300 neurons and the second hidden layer of 100 neurons. By increasing the first hidden layer to 1000 neurons and the second to 150, an error rate of 2.45% was reached.

Architectures that are similar to LeNet-5, namely its predecessors LeNet-1 as well as LeNet-4, achieved an error of 1.7% and 1.1%, respectively. By combining ("boosting") several instances of LeNet-4, the error rate got as low as 0.7%. This is even lower than what LeNet-5 showed. However, this boosted LeNet-4 about three times as much computation as LeNet-5. When used on input images that were distorted artificially, LeNet-5 performed better then all other methods.

5 Conclusion

Starting from a basic classifier, various concepts for pattern recognition were introduced. The derivation of gradient-descent learning was shown and applied to the multilayer perceptron, which yielded the backpropagation algorithm. These methods are widely used with neural networks and provide the foundation for convolutional neural networks.

Afterwords, the problems of the multilayer perceptron were discussed. It was shown, that the concept of convolutional neural networks and weight sharing not only reduces the need for computation, but also offers a satisfying degree of noise resistance and invariance to various forms of distortion.

LeNet-5, as an application of convolutional neural networks, was shown to outperform most other techniques. With an error rate as low as 0.95% it is still a little bit more error-prone than the boosted LeNet-4. However, the requirement for extensive computation in boosted LeNet-4 makes LeNet-5 the perfect candidate for hand-written digit recognition.

Bibliography

[1] LeCun Y, Bottou L, Bengio Y, Haffner P. *Gradient-Based Learning Applied to Document Recognition* in Proceedings of the IEEE, 1998

[2] LeCun Y, Bengio Y. *Convolutional Networks for Images, Speech, and Time-Series* in The Handbook of Brain Theory and Neural Networks. MIT Press 1995

[3] LeCun Y, Bengio Y. *Pattern Recognition and Neural Networks* in The Handbook of Brain Theory and Neural Networks. MIT Press 1995

[4] Haykin S. *Neural Networks: A Comprehensive Foundation, second edition.* Prentice Hall 1999. Chapter 4 Multilayer Perceptrons, pp. 156–255

[5] Schwenker F, Strey A. *Einführung in die Neuroinformatik* (lecture notes). University of Ulm, Germany 2005–2006

[6] Negnevitsky M. *Artificial Intelligence: A Guide to Intelligent Systems.* Addison Wesley 2002. Chapter 6 Artificial Neural Networks, pp. 163–186