An Analysis of Deep Learning Architectures inspired by Neuroscience

Moritz August
Department for Informatics, Technische Universitaet Muenchen

Abstract

Artificial neural networks were inspired by neuroscientific insights from the beginning. In this report, convolutional neural networks and two novel methods are presented and reported experiments repeated. Further experiments are conducted and ways to combine those methods are evaluated. While the results of reported experiments are mostly verified, the combination of the methods proves to be more difficult. This is a possible direction of further research.

1. Introduction

The rise of artificial neural networks (4) in the past decade has led to a high acceleration of research activity in the field. Since artificial neural networks are designed to resemble information processing structures in the brain, techniques which try to integrate neuroscientific ideas into the framework of deep learning are investigated. Convolutional neural networks (called CNNs) are a prominent example of such methods. They were originally proposed in (1) in 1998 and have since been successfully put to use in tasks such as image classification both in industry and research (5).

This work focuses on CNNs and two recent publications (2) (3) motivated by neuroscientific findings and tries both to evaluate the usefulness of these methods and to investigate the feasibility of their combination. In the following, this work will first introduce CNNs and describe the experiments conducted with them in Section 2. In Section 3, a technique called local winner-takes-all (2) will be presented together with the results achieved in experiments. Section 4 will then explain a related idea by the name of channel-out (3) and present results of the experiments run as well. This work will then end with a final discussion.

2. The Multilayer Perceptron

The methods presented in the following are all based on or extend the most simple deep learning architecture, the so-called multilayer perceptron (called MLP). To make it possible for the reader to see the bigger picture, the MLP model and its training process will hence be briefly revisited in this section.

A MLP is a function $f$ that is the composition of multiple functions $f_i$ given by

$$f_i = a_i(f_{i-1}W_i^T + b_i)$$

with $f_{i-1}$ being the output of the lower layer, $W_i$ being the matrix of the weights and $b_i$ being a bias constant vector. Note that $f_0 = X$ where $X$ is the input matrix being designed such that each row contains a sample. The function $a_i$ is a so-called activation function and is in most cases used to add nonlinearity to the affine transformation. Popular choices for $a_i$ are the logistic sigmoid function given by

$$a_i(x) = \frac{1}{1 + e^{-x}},$$

tanh or the rectifier function given by

$$a_i(x) = \max(0, x).$$

The $f_i$ are called the layers of the MLP.

To determine the quality of a model with respect to some problem, an error function $E$ is used. In a supervised training scenario as it is assumed in this work, $E$ measures the distance between the predictions of $f$ and the correct labels $z_j$ of the training samples $x_j$. A general formulation of $E$ is then given by

$$E(W) = \sum_{(x_j, z_j) \in D} d(z_j, f(W, x_j))$$

with $D$ being the dataset, $d$ being some distance function and $W$ being the matrix $(W_1, ..., W_n)$, $n$ denoting the number of layers. Note that in this notation, the biases $b_i$ are integrated into $W_i$.

To train an MLP $f$ to minimize $E$, the gradient $\nabla E$ is needed. For an MLP, this gradient can be computed by using back-propagation (6). In the simplest case, the weights
$W_t$ of the MLP at iteration $t$ of the training are then updated by

$$W_{t+1} = W_t - \alpha \nabla E$$

with $\alpha$ being the learning rate. This update rule is called (steepest) gradient descent.

3. Convolutional Neural Networks

This section will first explain the concept of CNNs and the implementation chosen in this project. Subsequently, the experiments conducted with CNNs will be presented.

3.1. The Concept

CNNs were proposed in (1) as a way of replacing pattern recognition methods for images using handcrafted features by techniques which are able to learn own representations from data. Thereby, another important goal was to achieve robustness against shifting, distortion and scaling of objects with only requiring minimal preprocessing of the images. To that end, four concepts were leveraged. Local receptive fields describe the idea of only allowing each neuron to have a specific part of an image as input e.g., a 5x5 pixel-field. This has two major advantages. Firstly, the total number of weights of the network is lowered, which induces both a lower capacity and performance benefits. Secondly, each neuron is able to learn distinctive local features in its receptive field.

Since features that are sensible in one part of an image might very well also be useful in another, LeCun et al. introduced the idea of shared weights. This term describes the concept of setting the weights of a group of neurons to be the same, i.e. let them share the weights, and by that make them learn to detect exactly the same feature. In CNNs, the receptive fields of such a group of neurons are then arranged such that neighbouring neurons have neighbouring receptive fields in the input layer, which my also overlap. This principle is illustrated in Figure 1 for one-dimensional input. For images, the receptive fields can be viewed as planes covering the image. A group of neurons organised in that way is called a feature map. This design has again some advantages: If an image is shifted or distorted and the location of the detected features changes, the feature map transforms in the same way as the features are now detected by other neurons corresponding to the areas of the image the features now lie in. This is a key factor for the robustness of the model. Furthermore, the sharing of the weights reduces the overall number of weights that have to be trained and can be regarded as a kind of regularisation. Feature maps can be seen as a convolution with the weights of the neurons in that feature map being the kernel. This gives rise to the definition: Convolutional Neural Network.

To classify images, only one feature per layer will not suffice in most cases. Hence, in a CNN, one layer comprises multiple feature maps each detecting a different feature. Such a layer is called convolutional layer. This effectively introduces a new function $f_{conv,i}$ to the MLP framework that can replace any function $f_i$ of a given architecture $f$.

LeCun et al. argue that the exact position of a feature becomes of less importance, once it is detected. Even more so, since the exact position of a feature might differ between samples of a class, depending on the precise location might even be harmful. To achieve a higher degree of robustness against this phenomenon, sub-sampling layers are introduced. In a sub-sampling layer, for each feature map in the previous layer there exists a corresponding feature map. The neurons of the sub-sampling feature maps compute the average of the inputs, multiply them with a trainable coefficient, add a trainable bias and apply an activation function to the result. It is worth noting, that the local receptive fields of the neurons in the sub-sampling layer do not overlap. This operation reduces the resolution of the feature maps and thereby counteracts the problem mentioned above. Instead of averaging the inputs, one might also employ the max pooling function. Like the convolutional layers, the sub-sampling layers extend the MLP framework by a function $f_{sub,i}$.

In a typical CNN architecture, a convolutional layer is followed by a sub-sampling layer and these pairs of layers are then stacked upon another. This translates to an alternating composition of $f_{conv,i}$ and $f_{sub,i}$ in a model $f$. For convo-
An Analysis of Deep Learning Architectures inspired by Neuroscience

Table 1. The results of the CNN for the reference and self-written implementation.

<table>
<thead>
<tr>
<th></th>
<th>Error on Test Set in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference Implementation</td>
<td>0.92</td>
</tr>
<tr>
<td>Own Implementation</td>
<td>0.94</td>
</tr>
</tbody>
</table>

volutional layers following sub-sampling layers, each neuron of a feature map receives the corresponding part of all the feature maps in the layer below as input and can hence build higher-level features from the more basic ones. This is depicted in Figure 2. Typically, the number of feature maps is raised for higher layers to compensate for the loss in resolution due to the sub-sampling layers. In most cases, a few fully-connected layers without any weight-sharing follow the convolutional and subsampling layers. A CNN model can then be trained as it was described in Section 2.

3.2. Implementation

The implementation of the CNN model was done using Python, Theano (8) and NumPy. The Code was written using and contributing to the Biomimetic Robotics and Machine Learning Lab’s deep learning library. The Theano tutorial for CNNs was used as a reference. It is noteworthy that, following the tutorial, instead of the averaging described in (1) the max pooling operation was used in the sub-sampling layers due to its lower runtime. These layers are referred to as max pooling layers from now on.

3.3. Verification

In the Theano tutorial for CNNs, an experiment was conducted on the MNIST dataset. To verify the correctness of the implementation, this experiment was repeated with the code written in the context of this work.

The model used for this experiment was a CNN with two convolutional layers with 20 and 50 feature maps respectively, each followed by a max pooling layer with a pooling size of 2x2. The size of the receptive fields was 5x5 for all feature maps in both layers and the activation functions were chosen to be the tanh function. These layers were followed by a fully connected layer with 500 units using the tanh activation function as well. The output layer was chosen to be a 10-unit layer with tanh output function and the error function used was the negative cross-entropy, as it is common for experiments on MNIST.

The results are stated in Table 1. As it can be seen, the result for the implementation written by the author was on par with the one reported for the reference implementation and hence the verification was successful.

3.4. Applying CNNs to time-series data

The advantages and disadvantages of using CNNs for image classification and similar tasks have already been studied extensively in the past. However, another interesting question is how well CNNs are suited for problems in other domains (9). One domain that comes to mind in particular is the domain of time-series data. As CNNs are designed to be able to detect a feature at differing positions in the input, the question arises, how well this might be applicable to temporal features rather than spatial ones.

To investigate this question, different CNN architectures were tested on an EMG dataset describing the movement of an arm over time. The specific task in this case was to determine the position of a marker on the arm in three-dimensional Euclidean space given the measurements of 12 EMG sensors also attached to the arm. The error function was hence chosen to be the squared error given by

$$E(w) = \sum_{(x_j, z_j) \in D} (z_j - f(w, x_j))^2$$

and the data-samples were given to the models as “images” of height 1 with 12 channels, a channel hereby denoting the measurements of a specific marker. To have a baseline for the experiments, results obtained at this task by a recurrent neural net were provided by Justin Bayer2.

After a few initial experiments I found that the CNNs made heavily oscillating predictions as is shown in Figure 3. To solve this problem, different pooling-sizes, activation functions, sizes of the receptive fields and $L_2$ regularization were tried out for several different architectures. Furthermore, the preprocessing of the data via ZCA was tested. While the variation of the aforementioned parameters had very little effect on the quality of the predictions, ZCA worsened the problem by not reducing the oscillation and

1http://deeplearning.net/tutorial/lenet.html

2Contact: bayer.justin@googlemail.com
Table 2. The error of a recurrent architecture and two exemplary CNN architectures on the test set for the x-, y- and z-coordinates. The best values are printed bold.

<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN</td>
<td>0.0043</td>
<td>0.0052</td>
<td>0.0120</td>
</tr>
<tr>
<td>CNN1</td>
<td>0.0169</td>
<td>0.0150</td>
<td>0.0319</td>
</tr>
<tr>
<td>CNN2</td>
<td>0.0067</td>
<td><strong>0.0048</strong></td>
<td><strong>0.0092</strong></td>
</tr>
</tbody>
</table>

simultaneously increasing the error on the test set significantly. In Table 2, the results for the following selected architectures are shown:

**RNN** The recurrent neural network used as baseline

**CNN1** 1 convolutional layer with 32 feature maps, 1 MLP layer with 16 units, ZCA was used for training

**CNN2** 1 convolutional layer with 32 feature maps, 2 MLP layers with 64, 64 units

The CNN architectures had an output layer with three units and the rectifier function was used for all hidden layers and the identity function for the output layer. For the sake of brevity, the remaining parameters are not listed. They were chosen such that they yielded the lowest oscillation of the predictions at visual inspection.

As can be seen in Table 2, the results achieved by the CNN2 were close to and in parts even superior to the results of the recurrent neural net.

4. Compete to Compute

In this section, the local winner-takes-all activation (from now on called LWTA) (2) function will be presented, followed by a brief note on the implementation done in the context of this project. Subsequently, the experiments conducted will be described.

4.1. Concept

Inspired by the fact that spatially close neurons in some regions of the brain exhibit competitive behaviour, Srivastava et al. propose a novel kind of activation function which leverages local competition among the units in a layer. This function can be seen as an alternative choice for the activation functions $a_i$ introduced in Section 2. For a given artificial neural network, the units are divided into $B$ blocks that are arranged in layers and contain $n$ units each. For every block $b_i$, $i \in 1, \ldots, B$ and unit $u_j$, $j \in 1, \ldots, n$ an output vector $y_i$ is computed with

$$y_i^j = g(h_i^1, h_i^2, \ldots, h_i^n),$$

$g()$ being the competition function and the $h_i^j$ being the activations of the corresponding units. Although many competition functions are possible in theory, the hard winner-takes-all function is chosen in (2) and given by

$$y_i^j = \begin{cases} h_i^j & \text{if } h_i^j \geq h_k^j, k \in 1, \ldots, n \\ 0 & \text{otherwise} \end{cases}$$

The activation function is thereby defined to be the identity function. If ties exist, they are broken by choosing the activation with the lower index.

This means that LWTA sets the activations of all units in a block to zero, except for the one with the highest activation. Therefore, during training only the weights of the winning units are updated in a backpropagation pass. As there are only $B$ active neurons given an input, training on a dataset involves training $nB$ different models with partly shared weights together with the information which model should be used for which kind of input. As it is pointed out in (2), this might be beneficial in the context of learning on data with different sub-distributions. The concept of having exponentially many models within one network architecture is illustrated in Figure 4.

Although LWTA resembles techniques like Maxout or Dropout (10) at first glance, there are important differences between these concepts. In contrast to Maxout, LWTA does not reduce the number of units in its blocks but only sets the activations of the losing units to zero. While in LWTA the competition of units in a block is the non-linear element of the model and hence key to its performance, in Dropout the deactivation of units is used as a model-averaging and regularization technique to make single units, each possibly having non-linear activation functions, more independent from other units.

According to (2), the training of different models for differ-
ent subclasses of patterns makes models using LWTA also more robust against the so-called catastrophic forgetting. That means, that if a model is first trained on a dataset $A$ and on a dataset $B$ afterwards, its performance on $A$ will not decrease as fast using LWTA than with other architectures.

4.2. Implementation

The implementation of LWTA was as well done in the context of the lab’s library using Python and Theano. LWTA was implemented as a Theano expression using the $\max$ and $\arg\max$ methods provided by the framework as they implicitly implement the otherwise inefficient tie-breaking.

4.3. Experiments

In (2), the results of two experiments on the permutation invariant MNIST dataset are presented. In this scenario, 2D information is disregarded so that every image is only seen as a vector of 784 dimensions. The goal of these experiments as stated by the authors was to evaluate the performance of LWTA as only non-linearity in the models. Hence, for all LWTA-layers the identity was used as activation function and no preprocessing was applied to the data, except for the normalization of the input values to $[0, 1]$. For these experiments, a block size of 2 was used which will be referred to for the remainder of this section as LWTA-2.

In the first experiment, a model comprising three LWTA-2 layers with 500 blocks each and a softmax output layer with 10 units was trained until convergence on the merged training and validation set. For the second one, a CNN with two convolutional layers, two LWTA-2 layers and a softmax output layer was assessed. The convolutional layers thereby were employing the rectifier activation function and had 16 and 32 feature maps with a filter shape of 7x7 and 6x6 respectively. Max pooling was performed after both convolutional layers with a pooling size of 2. Each of the LWTA-2 layers comprised 64 blocks. To verify the results reported for these experiments and also the correctness of the implementation, these experiments were repeated in the context of this project. However, for the first experiment the model used was chosen to have only two LWTA-2 layers with 500 blocks to ease the training process. The results for the first experiment are shown in Table 3. As can be seen, the error of the model implemented in the context of this work is close to the one reported in (2) despite missing one layer. It was therefore concluded that the implementation of LWTA is correct. Furthermore, since the results achieved by the models without any data augmentation are well above average, the usefulness of LWTA as activation function was regarded as verified.

In the second experiment, the training of the CNN presented in (2) proved to be difficult. Initially, stochastic gradient descent and rmsprop were used with manually adjusted values for batch size, random initialization of the weights, regularization coefficients, step rate, etc.. As this did not yield the reported results, the authors of (2) were contacted to find out about the correct optimization parameters. However, also these parameters turned out to lead to inferior results. To overcome this problem, an exhaustive random search in the optimization parameter space was done but to no avail. The test set error as reported in (2) and the best error achieved during the experiments conducted by the author are shown in Table 4. With 1.03%, the error of the best weights found for the model is almost twice as high as the reported error. This is especially interesting, since both the implementation of CNNs and LWTA were verified independently before and a great variety of optimization parameter vectors were tested. Further research in this direction might therefore be beneficial.

5. Channel-out Networks

This section will present a very similar approach to LWTA called channel-out that is based on a more general paradigm. This is then followed by a short description of the implementation and the presentation of the conducted experiments. The section will close with a few notes on the implementation of channel-out for feature maps in CNNs.

5.1. The Concept

Like Srivastava et al. the authors of (3) are motivated by a neuroscientific finding, namely the fact that the pathway a signal takes in the brain is more important than its shape. To apply this insight to the domain of artificial neural nets, they define a mathematical framework for the recently introduced kind of activation function that is equivalent to the one reported in Section 4. The definitions are therefore

| Table 3. The test set error on the MNIST permutation invariant dataset in % for the LWTA-2 model presented in (2) and the one implemented during this project. |
|--------------------------|------------------|
| LWTA-2 2 layers          | 1.32             |

| Table 4. The test set error on the MNIST permutation invariant dataset in % for the CNN LWTA-2 model as reported in (2) and as achieved during this project. |
|--------------------------|------------------|
| Reference CNN LWTA-2    | 0.57             |
| Own CNN LWTA-2           | 1.03             |
omitted and the terms of Section 4 will be used if necessary. Examples for possible competition functions are given by \(\max()\), \(\min()\), \(\text{median}()\). Also the \(\text{top } k\) function that returns the \(k\) largest elements of a set could be used. It is pointed out that in principle, every function can be used as long as it meets the following requirements:

- The function must be piece-wise constant with constant regions of sufficient size to make it robust against noise
- The preimage for each value must be of similar size to ensure every output value is equally likely
- The evaluation of the function must be efficient in order for it to be scalable to real-word applications

It is furthermore assumed that all units in the hidden layers of a network employ the identity activation function. Networks that make use of this architecture are called channel-out networks.

Wang et al. argue that this class of activation functions gives rise to a new information encoding behaviour. Instead of encoding different parts of information within each subnetwork as neural nets with commonly used activation functions do, channel-out networks encode all information of a specific input class into one or only a few paths through the network. Even more so, by design they are also able to activate the corresponding pathways for a given input at prediction time. This phenomenon was also hinted at in (2). Wang et al. see this as a new paradigm for neural network architectures and refer to it by the term sparse path-way coding.

The learning behaviour of channel-out networks is further analyzed and divided into two classes: activation tuning and pathway switching. Activation tuning happens when the update of the weights is not significant enough to change the output, i.e. pathway selection, of a block. Pathway switching denotes the contrasting case. As it is pointed out in (3), if pathway switching occurs in a block this has a strong effect on the path above this block due to the specific output paths for each unit in a block. According to the authors, this will lead to the learning of a very different path if the old one proved to be ineffective, which is a desirable learning behaviour.

In addition to the aforementioned advantages it is claimed that channel-out networks perform especially well when combined with Dropout as the two approaches are viewed as complementary. Dropout aims to encode the complete information of the training data into all subnetworks and thereby increases the robustness. Channel-out networks try to encode the information of different patterns in the input data within different subnetworks or pathways to increase the inference capability. The latter approach could however lead to the effect of not utilizing all of the capacity of a network while the former technique in contrast could cause interference problems when very different patterns are tried to be encoded into the same low-capacity subnetwork.

Thus, in extreme cases, Dropout can lead to under-fitting while channel-out tends to over-fit. Wang et al. therefore argue both approaches naturally complement each other and lead to superior performance. The different information encoding behaviours of Dropout, channel-out and standard deep architectures are illustrated in Figure 5.

5.2. Implementation

Since LWTA is equivalent to channel-out using the \(\max()\) function, no further implementation was needed in this case. A very interesting function to use with channel-out seemed to be \(\text{top } k\) although it appears inefficient at first glance due to the need to use a loop with \(k\) iterations. However, an implementation was found that only needs a loop at build time, generating an efficiently evaluateable Theano expression.

5.3. Experiments

As (3) was only discovered by the author after LWTA had already been implemented and tested, an evaluation of channel-out and its aforementioned advantages would have been redundant. The relation between channel-out and dropout as explained above seemed to be the more interesting aspect and was therefore examined.

To have a baseline, a network with two fully-connected hidden layers and 512 neurons each was trained on MNIST with Dropout. The rectifier function was set to be the activation function for all hidden units and the standard 10-
Table 5. The test set error on the MNIST in % for a standard deep architecture and two channel-out networks.

<table>
<thead>
<tr>
<th></th>
<th>Error on Test Set in %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dropout + Rectifier</td>
<td>1.63</td>
</tr>
<tr>
<td>Dropout + Channel-out Max</td>
<td>1.58</td>
</tr>
<tr>
<td>Dropout + Channel-out Top 2</td>
<td>1.52</td>
</tr>
</tbody>
</table>

unit output layer with softmax was used. The dropout-probability for the input dimensions was chosen to be 0.2 and set to 0.5 for the hidden units. The same architecture was then trained again with Dropout, only that this time instead of the rectifier function a channel-out architecture using $\text{max}()$ as selection function was used. The blocksize was chosen to be 2 in this network. In a third iteration, the network was trained yet again, this time as a channel-out network using the $\text{top} \ 2$ function on blocks of size 4.

The best error obtained for each of the models is shown in Table 5. It can be observed that combining Dropout with channel-out indeed reduced the error significantly, especially in the case of the channel-out $\text{top} \ 2$ architecture. This gives reason to assume that the hypothesis of Wang et al. is correct. It is worth noting, that the goal of these experiments was not to obtain the best possible results and therefore no attempt was made to achieve lower errors by adapting the Dropout probabilities.

5.4. Applying sparse pathways to CNNs

During the work on channel-out architectures, the idea of applying channel-out within and between feature maps in CNNs was examined, too. A possible implementation was found but since it was not efficient enough and most likely no significantly faster implementation is possible in Theano, the idea was abandoned.

6. Discussion

This work was concerned with the evaluation of deep learning architectures inspired by neuroscientific findings. As it was shown above, these techniques indeed contribute to the state-of-the-art in deep learning and are well worth further investigations. It is important to note however, that in order to obtain well understood and reliable models a sound mathematical framework has to be devised first. If this is done as it was in (2) and (3), neuroscientific insights can beneficially be applied to deep learning.

The combination of CNNs with the two presented techniques proved to be more difficult than expected but is clearly possible. Further research in this direction might yield better solutions for this problem and could also solve the problems discovered when applying CNNs to time-series data.

References


