Artificial neural networks - Exercise session 4

Deep Learning: Stacked Autoencoders and Convolutional Neural Networks

2017-2018

1 Stacked Autoencoders

1.1 Introduction

An *autoencoder* neural network is an unsupervised learning algorithm that applies backpropagation, setting the target values to be equal to the inputs. I.e., it uses $y(i) = x(i)$ [1].
Figure 1 shows an example of an autoencoder. The autoencoder tries to learn a function $h_{W,b}(x) \approx x$. In other words, it is trying to learn an approximation to the identity function, so as to output $\hat{x}$ that is similar to x. The identity function seems a particularly trivial function to be trying to learn; but by placing constraints on the network, such as by limiting the number of hidden units (called a *sparse* autoencoder), we can discover interesting structure about the data. In fact, this simple autoencoder often ends up learning a low-dimensional representation similar to PCAs.

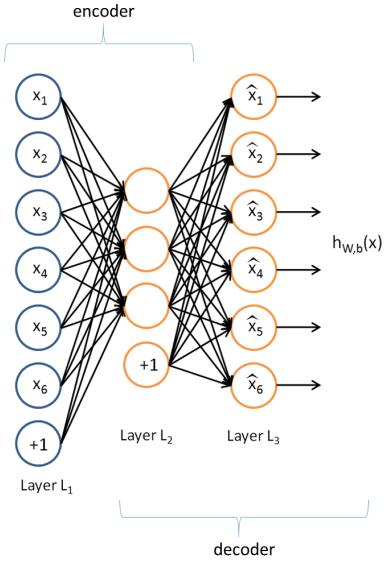

Figure 1: An example of an autoencoder. The part from the input layer to the hidden layer is called the *encoder*. The part from the hidden layer to the output layer is called the *decoder*.

An example of the use of an Autoencoder can be found in [1].

A *stacked autoencoder* is a neural network consisting of multiple layers of sparse autoencoders in which the outputs of each layer are wired to the inputs of the successive layer [2]. The information of interest is contained within the deepest layer of hidden units. This vector gives us a representation of the input in terms of higher-order features. For the use of classification, the common practice is to discard the ”decoding” layers of the stacked autoencoder and link the last hidden layer to the a classifier, as depicted in Figure 2.

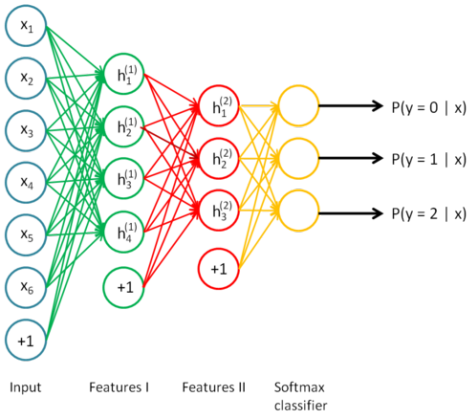

Figure 2: An example of a stacked autoencoder used for classification.

A good way to obtain good parameters for a stacked autoencoder is to use greedy layer-wise training. So we first train the first autoencoder on the raw input to obtain the weights and biases from the input to the first hidden layer. Then we use this hidden layer as input to train the second autoencoder to obtain the weights and biases from the first to the second hidden layer. Repeat for subsequent layers, using the output of each layer as input for the subsequent layer. This method trains the parameters of each layer individually while freezing parameters for the remainder of the model. To produce better results, after this phase of training is complete, fine-tuning using backpropagation can be used to improve the results by tuning the parameters of all layers at the same time.

A concrete example of a stacked autoencoder can be found in [2].

1.2 Exercise

Run the script `StackedAutoEncodersDigitClassification.m` [3] from Toledo, try to understand what is happening. Use the script `DigitClassification.m` from Toledo to investigate the different parameters (`MaxEpochs`, number of hidden units in each layer, number of layers) and to compare the performance of the Stacked Autoencoder to a normal multilayer neural network. Do not forget to comment the command `rng('default')` when you want to average the results over multiple runs.

Are you able to obtain a better result with different parameters (wrt to the default ones)? How many layers do you need to achieve a better performance than with a normal neural network? What can you tell about the effect of finetuning?

2 Convolutional Neural Networks

2.1 Introduction

Convolutional Neural Networks (CNN) is a deep learning technique that uses the concept of *local connectivity*. In a normal multilayer neural network all nodes from subsequent layers are connected, we call these models *fully connected*. The idea is that in a lot of datasets, points that are close to each other, are likely to be a lot more connected than points that are further away. For example, in image datasets where the datapoints represent pixels. Pixels that are close are likely to represent the same part of the image, while pixels that are further away can represent different parts.
CNN's are explained in the Stanford tutorial [4], and also in the following YouTube video trough an example [5].

2.2 Exercise

Run the script `CNNex.m` [6] from Toledo, try to understand what is happening[1] .

Take a look at the layers of the downloaded CNN and answer the following questions:

- Take a look at the first convolutional layer (layer 2) and at the dimension of the weights (`size(convnet.Layers(2).Weights)`). If you think back at what you saw in class and/or in [5], what do these weights represent?
- Inspect layers 1 to 5. If you know that a ReLU and a Cross Channel Normalization layer do not affect the dimension of the input, what is the dimension of the input at the start of layer 6 and why?
- What is the final dimension of the problem (i.e. the number of neurons used for the final classification task)? How does this compare with the initial dimension?

The script `CNNDigits.m` [7] runs a small CNN on the handwritten digits dataset. Use this script to investigate some CNN architectures. Try out some different amount of layers, combinations of different kinds of layers, dimensions of the weights, etc. Discuss your results. Be aware that some architectures will take a long time to train!

3 Report

Write a report of maximum 2 pages (including text and figures) to discuss:

- Digit Classification with Stacked Autoencoders and CNN.
- The answers to the three questions in Section 2.2.

References

[1] UFLDL Tutorial: Autoencoders
`http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders`

[2] UFLDL Tutorial: Stacked Autoencoders
`http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders`

[3] MathWorks documentation: Train Stacked Autoencoders for Image Classification
`https://nl.mathworks.com/help/nnet/examples/training-a-deep-neural-network-for-digit-classification.html?requestedDomain=www.mathworks.com`

[4] UFLDL Tutorial: Convolutional Neural Network
`http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork`

[5] YouTube: How Convolutional Neural Networks work
`https://www.youtube.com/watch?v=FmpDIaiMIeA&list=LLoyD7vXkP56-wnw6rR18S6w&index=1`

[6] MathWorks documentation: Image Category Classification Using Deep Learning
`https://nl.mathworks.com/help/vision/examples/image-category-classification-using-deep-learning.html`

[7] MathWorks documentation: Create Simple Deep Learning Network for Classification
`https://nl.mathworks.com/help/nnet/examples/create-simple-deep-learning-network-for-classification.html`

[1] Since we don not have access to a CUDA-capable GPU on the computers we can not train a large CNN in class. If you are interested in CNNs and you have acces to a GPU for computing you may try the full demo [6] at home (note: this is not necessary for the report).