Bachelor practical: Report

Marco Unternährer

December 13, 2015

1 Introduction

Training a spiking neural network is a challenging task. Neuromorphic vision researchers have to create spiking datasets since there is a lack of such datasets publicly available . One method for converting static image datasets into neuromorphic vision datasets was proposed by moving a sensor in front of an image [9].

Another approach for training a spiking neural network was developed by researchers at the Institute of Neuroinformatics [5]. A non-spiking neural network is trained by using a regular static image dataset like MNIST or Caltech-101. The resulting weights of the trained neural network can then be used to create a spiking neural network. One constraint for this to work is that units must have zero bias.

The task for this bachelor practical was to train a convolutional neural network on the Caltech-101 dataset.

2 Image Classification on Caltech-101

This section will give a short introduction to the Caltech-101 dataset, the used methods and techniques.

2.1 Caltech-101

[6]

2.2 Convolutional Neural Network

A convolutional neural network (CNN) is a type of feed-forward artificial neural network. CNNs are biologically-inspired variants of multilayer perceptrons [1]. From Hubel and Wiesel's early work on the cat's visual cortex [7], it is known that the visual cortex contains a complex arrangement of cells [1]. These cells are sensitive to small sub-regions of the visual field, called a receptive field [1]. The sub-regions are tiled to cover the entire visual

field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images. Additionally, two basic cell types have been identified: Simple cells respond maximally to specific edge-like patterns within their receptive field. Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern [1].

Like their biological relatives, units in a CNN take advantage of the 2D structure of the input data (images or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features [2].

2.3 Batch Normalization

bn [8]

2.4 Cyclical Learning Rates

In [3], Bengio says that the learning rate is the most important hyper-parameter to optimize and if "there is only time to optimize one hyper-parameter and one uses stochastic gradient descent, then this is the hyper-parameter that is worth tuning". It is well known that too small a learning rate will make a training algorithm converge slowly while too large a learning rate will make the training algorithm diverge [13]. Hence one must experiment with a variety of learning rates and schedules (i.e., the timing of learning rate changes) [10].

The conventional wisdom is that the learning rate should be a single value that monotonically decreases during the training [10]. However, Smith demonstrates the surprising phenomenon that increasing the learning rate is overall beneficial and thus proposes to let the global learning rate vary cyclically within a band of values rather than setting it to a fixed value [10].

3 Experimental Setup

3.1 Data Pre-processing & Data Augmentation

Since the heights and widths of the Caltech-101 images vary, all images were scaled to a common size. Like [9], each image was resized to be as large as possible while maintaining the original aspect ratio and ensuring that width does not exceed 240 pixels and height does not exceed 180 pixels.

After running a few experiments with a fairly deep architecture, it became clear, that the roughly 10'000 images of the Caltech-101 dataset were not enough to accomplish a decent test accuracy. Starting from the original dataset, data augmentation was pursued by randomly transform each picture ten times, and thus produce approximately 100'000 images.

Using an image data generator [4], we ended up with the following augmentation parameters:

- rotation: random with angles in the range 0° to 20°
- translation: random with shift between 0% and 20% of total width/height
- flipping: horizontal/vertical, yes or no (Bernoulli)
- zooming: random zoom axis by factor +/- 20%

The created images were then resized again, but with the difference that the images were padded with the edge pixels to fill the needed aspect ratio. The pixel values were normalized to the range of (0, 1) after loading the images. Data standardization, subtracting the mean and dividing by the standard deviation, has sped up the training and was also applied.

The generated images are available online and can be downloaded under [12].

3.2 Architecture

architectures inspired by Simonyan2015

constraints: 3 convolutional layers, 2 fc

Layer type	Parameters
convolution	128x5x5, stride 2x2
relu	
maxpool	2x2
convolution	256x3x3
relu	
maxpool	2x2
convolution	512x3x3
relu	
maxpool	2x2
full	1024
relu	
full	102
softmax	

Table 1: Base Network Architecture

Layer type	Parameters
convolution	128x5x5, stride 2x2
batch normalization	
relu	
maxpool	2x2
convolution	256x3x3
batch normalization	
relu	
maxpool	2x2
convolution	512x3x3
batch normalization	
relu	
maxpool	2x2
full	1024
relu	
full	102
softmax	

Table 2: Network Architecture with Batch Normalization

Layer type	Parameters
convolution	128x5x5, stride 2x2
relu	
maxpool	2x2
dropout	0.35
convolution	256x3x3
relu	
maxpool	2x2
dropout	0.35
convolution	512x3x3
relu	
maxpool	2x2
dropout	0.35
full	1024
relu	
dropout	0.5
full	102
softmax	

Table 3: Network Architecture without Batch Normalization

3.3 Training

train / test split 81378 train samples, 9104 validation samples, 10102 test samples -¿ train/test split saved to disk unlike MNIST, caltech-101 doesnt have a predefined split between train/test data

sgd with nesterov

learning rate schedule

3.4 Implementation Details

The neural network framework of choice was Keras [4]. Keras is a minimalist, highly modular neural networks library, written in Python and running on top of Theano. Being Python-based and especially developed with a focus on enabling fast experimentation, it was a good candidate for this project. Most importantly, Keras seemed to provide almost all needed functionality out of the box. Another relevant aspect, that lead to the decision to use Keras, was that model architectures are created with code. That made rapid prototyping and experimentation feasible.

The networks were trained on a system equipped with a single NVIDIA GTX 970 GPU, and training the best performing net took only ???? to converge.

The source code for the entire bachelor practical can be found at [11].

4 Results

w/o batch normalization, w/o data normalization with bn, with data normalization

5 Conclusion

enough data is important bn can speed up training time significantly

References

[1] Convolutional Neural Networks (LeNet).

[2] Convolutional Neural Network, 2015.

[3] Y. Bengio. Practical Recommendations for Gradient-Based Training of Deep Architectures. *arXiv*, 2012.

[4] F. Chollet. Keras.

[5] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-c. Liu, and M. Pfeiffer. Fast-Classifying, High-Accuracy Spiking Deep Networks Through Weight and Threshold Balancing.

[6] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, 2007.

[7] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, mar 1968.

[8] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Arxiv*, 2015.

[9] G. Orchard, A. Jayawant, G. Cohen, and N. Thakor. Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades. pages 1–15, 2015.

[10] L. N. Smith. No More Pesky Learning Rate Guessing Games. *Arxiv*, 2015.

[11] M. Unternährer. INI Caltech101 Code, 2015.

[12] M. Unternährer. INI Caltech101 Generated Images, 2015.

[13] M. D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv*, page 6, 2012.