Deep Neural Networks in Embedded Applications

How to develop intelligent systems

Andrés Mlinar, Embedded Trend GmbH

In recent years very deep artificial neural networks (DNNs) have been created that successfully solve many difficult problems. This opens up entire new areas of application. DNNs are a solution when there is a large amount of data showing the expected outcome for a given input. DNNs can be trained using this data to be able to give sensible answers when presented with new, never seen before, data.

These capabilities are also desirable in the embedded field, for example to develop intelligent sensors. However, the large computation capabilities necessary are a limiting factor. A solution is to acquire the data on the embedded device and send it for processing to a cloud service. This has several limitation like connectivity costs, latency and privacy.

The lecture describes a method by which deep neural networks can be deployed to embedded devices, using currently available processors, with limited processing power and limited available memory.

DNNs are a powerful computational mechanism modeled after the general principles observed in biological neural networks found in animals. A neural network is built out of simpler non-linear elements called neurons that are combined into larger structures. DNNs are organized into layers of neurons. Each layer contains one or more neurons that are connected to other neurons on the same or other layers. Typically all inputs to the network are connected to the input layer and the output from the DNN comes from the neurons in its output layer. Between the input and output neuronal layers there can be none, one or many hidden layers. What makes an artificial neural network "deep" is the use of several hidden layers (Figure *1*).

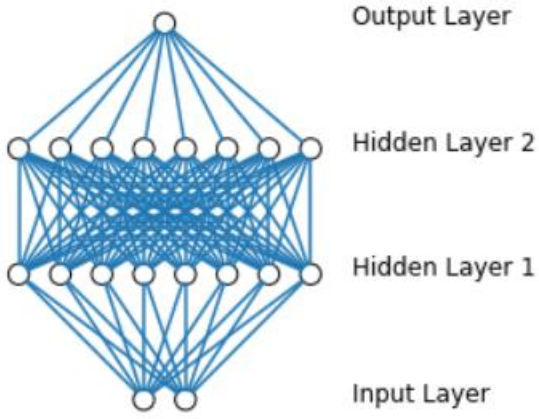

Figure 1: deep feed-forward neural network

There are different types of DNNs. The simplest form of DNN is a feed-forward network, where information flows in only one direction, from input to output. In computer vision convolutional neural networks (CNN) are common; these are a special type of feed-forward network, where layers of neurons that act as feature filters are convolved over the input, this has been shown to improve performance and reduce the memory footprint. Recurrent Neural Networks (RNN) are a class of neural networks in which neurons form cycles; these are used for processing sequences, like natural language, speech or video.

The main advantage of DNNs is that they can be used to solve difficult problems that have evaded hand-crafted solutions. This is due to the fact that DNNs are not manually programed, but they learn their parameter values by example. This process is known as supervised learning. The DNN is presented with many examples of input data where the solution is known. Then the values in the neural network are gradually adjusted for each example, making the parameters better approximate the expected result in each step (Schmidhuber, 2015). This process is repeated a very large number of times until the neural network is capable of giving good answers for the training data (Figure *2*: neural network accuracy improvement during trainingFigure *2*). Such network can then be used on new and never seen before data. This learning process is computationally very demanding. This is generally run on powerful computers with specialized hardware. Most of the training steps can be very efficiently implemented in graphical processing units (GPUs) that were initially designed to speed up 3D graphical software, as the bulk of the underlying mathematical operations are similar: matrix multiplications.

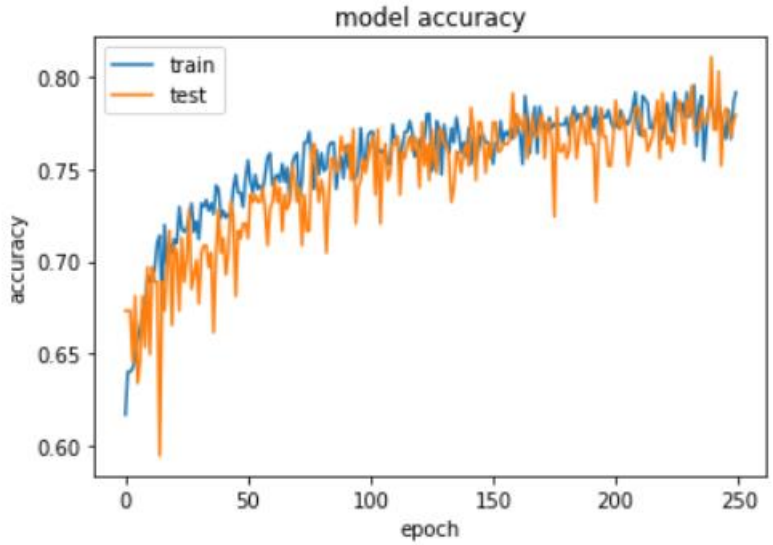

Figure 2: neural network accuracy improvement during training

Today most DNNs are deployed on server in the cloud or using specialized systems. In this hardware the computational capabilities and memory available are plentiful. Early DNN implementations have counted on these ample resources. As the field progressed it has been realized that it was possible to achieve similar results using smaller DNNs. Two of the most effective ways to achieve this are new network

architectures (Forrest N. Iandola, 2016) and network compression (Karen Ullrich, 2017). The initial drivers of these steps have been to improve performance and to lower power consumption. The same advances open the door to deploying DNNs in less capable hardware, as general embedded devices.

Several frameworks for the development and deployment of DNNs are available today. One of the earliest was Caffe, developed Yangqing Jia at UC Berkeley (Jia, 2014). Caffe has been branched by Facebook to make their own framework Caffe2 (Facebook, 2017). Google also developed its own framework TensorFlow (Google, 2015). Microsoft joined in the crowd with its own Cognitive Toolkit (Microsoft, 2016). There are several other software packages that can be used during neural network research, design and deployment.

One common trait of these frameworks is their reliance on a general purpose computer, with a general purpose operating system. This is good for cloud applications and also mobile, but excludes them from use in embedded systems, where only a bare metal environment or an RTOS are available. Their computational engines are also optimized for CPUs and GPUs used in server and workstations.

To deploy DNNs to embedded devices one approach that has been used is to communicate the data to a cloud based back-end to run the computationally demanding DNN. This approach has several problems. A communication link is necessary, increasing the cost of the unit. Connectivity is not always possible, as the device may need to operate in environments without network coverage. Power demands on the system that needs to keep a connection alive are also higher. Latency due to communication is increased. Probably most important of all, is that the data travels on the network and is used in a separate system, introducing issues of security, privacy and trust.

A much better approach is to run the neural network directly on the embedded device. The neural network has to first be designed and trained on a powerful system. Modern DNN architectures that are specifically tailored to reduce the number of parameters should be used, to reduce the necessary computation and memory demands on the embedded system. Once the DNN parameters have been trained, further work is performed to compress the network using different techniques.

When the DNN is ready to be deployed, the implementation on the embedded system should be done with care. No standard framework can be used, as they rely on OS, libraries and base software that are not available on the embedded device. This implementation should be stripped of all unnecessary software dependencies that were used during design and training. Of particular importance is the handling of memory, as dynamical allocation of large buffers is neither desirable nor possible. Computation should be adapted to the capabilities of the chip used, as the differences in performance between naïve software-loop based high level code and highly optimized hand-crafted assembler can be many hundred times.

Summary

DNNs are a new powerful software tool that is gaining prominence. They are used extensively on the cloud and powerful mobile devices, but the potential exist to use them also in mass market embedded device. With the necessary knowledge and techniques it is possible to achieve this goal.

List of abbreviations
DNN – Deep Neural Network
CNN – Convolutional Neural Network
RNN – Recurrent Neural Network
GPU – Graphical Processing Unit

List of Figures
Figure 1: deep feed-forward neural network
Figure 2: neural network accuracy improvement during training

Literature and References

Facebook. (2017). *Caffe2*. Retrieved from A New Lightweight, Modular, and Scalable Deep Learning Framework: https://caffe2.ai/

Forrest N. Iandola, S. H. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv:1602.07360*.

Google. (2015). *TensorFlow*. Retrieved from An open-source software library for Machine Intelligence: https://www.tensorflow.org/

Jia, Y. a. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. *arXiv preprint arXiv:1408.5093*. Retrieved from Caffe: http://caffe.berkeleyvision.org/

Karen Ullrich, E. M. (2017). Soft Weight-Sharing for Neural Network Compression. *arXiv:1702.04008*.

Microsoft. (2016). *MSCTK*. Retrieved from The Microsoft Cognitive Toolkit: https://www.microsoft.com/en-us/cognitive-toolkit/

Schmidhuber, J. (2015, January). Deep Learning in Neural Networks: An Overview. *Neural Networks*, pp. 85-117.

Author

Andrés Mlinar has more than 15 years of experience in embedded software, specializing in the areas of development tools, RTOS, middle-ware, drivers and system software. In the last few years he has moved his focus to the fields of machine learning and artificial intelligence.

Kontakt

Internet: www.embeddedtrend.de
Email: andres.mlinar@embeddedtrend.de