Neural Networks for fast sensor data processing in Laser Welding

Johannes Günther, Hao Shen, and Klaus Diepold

Dept. of Elecrical and Computer Engineering, Technische Universität München
80290 Munich, Germany
{johannes.guenther,hao.shen,kldi}@tum.de
http://www.ldv.ei.tum.de

Abstract. *To address the need for robust and fast representation, we introduce deep learning neural networks and parallel programming techniques for laser welding. In order to deal with high-dimensional data within real-time constraints, we use a deep autoencoder to extract robust, meaningful and low dimensional features. The implementation is then optimized, using parallel programming techniques and shown to perform within the real-time requirements for laser welding. The performance, in terms of reconstruction and capability for classification are later compared with features, extracted by the principal component analysis. The neural network demonstrates to extract more robust and meaningful features, compared to a PCA.*

Keywords: Laser welding, deep learning, fast sensor data processing

1 Introduction

To fit the upcoming requirements of modern industry [1], it is necessary to empower systems to learn and improve their own performance in order to be able to ensure flexible and autonomous systems. Those systems would be able to deal with changing process and environmental conditions, as they often occur in industry. Being inspired by the human capability of learning and adapting, a whole branch of research has centered around the challenge of providing machines with intelligence. This research is summarized with the term machine learning. By applying algorithms of machine learning, important concepts of intelligence, like perception, reasoning, learning and planning can be imitated.
The basic for learning and autonomy is the ability for perception and abstraction. In industrial systems, the pure observation is usually done by sensors, e.g. cameras, and the process of abstraction is provided by data processing. One way of processing images is to extract meaningful features. This is comparable to the term of abstraction. The transformation into features has two major advantages: First, features are much less data, so they can be processed in less time. Second, extracting features can make the representation much more robust. As during this process, fluctuations and disturbances in the process are filtered out, only the truly relevant information is retained in the features.

One particular process that can benefit from these abilities is laser welding. Due to its dynamics and uncertainty, inherent to the process, it can not be controlled using a model. Therefore the application of machine intelligence is a feasible approach. Especially the usage of neural networks has been topic of multiple investigations. However, these approaches either use neural networks for classification [2], [3] or directly for control [4], [5] and not for features extraction. Additionally, the networks, used so far only consist of a small number of neurons, while the neural network in this paper has several thousand neurons.
A technique that has shown its capability for robust feature extraction is deep learning, especially autoencoders [7]. However, large neural networks do require a huge amount of computation. One approach to reduce the computational time is to parallelize the algorithm in order to perform multiple computational steps at the same time.
The combination of both approaches, neural networks and parallelization, promises to provide a solution for real time feature extraction, using neural networks. This is described in the remains of this paper as follows. Section 2 provides knowledge about representation learning and neural networks in detail. Section 3 focuses on the preprocessing, the design of the autoencoder and the parallelization details. The experiments are described and evaluated in Section 4 and discussed in Section 5. The paper will be concluded in Section 6.

2 Representation Learning

It is well known that the performance of machine learning algorithms is mostly dependent on the data. But not only the amount of data is crucial, but also the representation. Different representations might capture different hidden information and dependencies in the data. Being able to extract this data automatically is an important step towards autonomous artificial intelligence [6].

2.1 Principal Component Analysis

The principal component analysis is arguably the most popular method, which transforms the original data into a set of data representations, often in a lower dimensional space compared to the original data space. By finding an orthogonal transformation, original data, often assumed to be statistically correlated, is transformed into a set of statistically uncorrelated data representations. The new set of data representations are referred to as the principal components. When the number of uncorrelated components is smaller than the dimension of the original data space, PCA serves as a tool of dimensionality reduction, with a great simplicity in terms of both computation and reconstruction of the original data.
PCA has been successfully applied to a broad variety of applications, such as medicine [9], chemistry [10], face recognition [11] and also laser welding [12].

2.2 Artificial Neural Networks

The idea of artificial neural networks (ANN) to imitate the human capability for recognition and generalization was introduced in 1958 [13]. The perceptron is the smallest version of an ANN, consisting only of a single layer of neurons. A neuron is modeled as a function $\sigma(\cdot)$, that takes an arbitrary number of inputs. These inputs are multiplied by corresponding weights, summed up and shifted by a bias. The activation or output of the neuron is then calculated by passing the sum through the nonlinear function, called activation function. This can be described by the following general equation:

$$y_i = \sigma(\sum_{j=1}^{N} W_{j,i} x_j + b_i) \tag{1}$$

where y_i is the output of the i-th neuron and x_j is the j-th input of that neuron. $W_{j,i}$ is the weight for the j-th input of the i-th neuron and b_i is the bias for the i-th neuron. N is equal to the total number of inputs for that neuron. This is illustrated in figure 1.

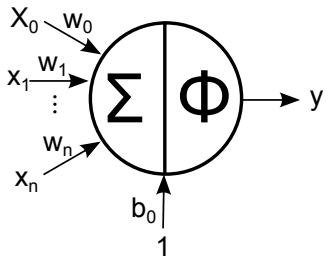

Fig. 1. A single neuron

However, the simple perceptron has limitations. Its capabilities are constrained to linear calculations. To overcome these limitations, several perceptrons can be connected in a way, that the output of one perceptron is the input for the next one. This is called a multi-layer perceptron (MLP) or feed-forward network. A general MLP can have an arbitrary number of neurons and layers. By introducing more than one layer, the network is able to learn hidden representations for the input. To update the weights of the hidden representations the backpropagation algorithm was invented [14]. It passes the error backwards through the MLP and therefore allows the hidden layers to adapt their weights in order to improve the performance[15].

2.3 Deep Autoencoder

The autoencoder was introduced by Hinton and Salakhutdinov in 2006 as a nonlinear generalization of the PCA [16]. An autoencoder uses a MLP to encode

the input into a lower dimensional representation, according to equation 2.

$$h = \sigma(\sum_{j=1}^{N} W_j x_j + b) \tag{2}$$

This representation is later decoded by a similar MLP to reconstruct the original input.

$$y = \sigma(\sum_{j=1}^{N} W_j^\top x_j + b') \tag{3}$$

As it is difficult to optimize the weights in a nonlinear autoencoder with several layers, the learning process is done layer-wise. For this purpose, first an autoencoder with only one hidden layer is trained. After the training is completed, the weights are fixed and another layer is stacked on top. The representation of the first, already learned, autoencoder serves as input for the second autoencoder. Using this technique, it is possible to learn a better representation than by learning the whole autoencoder at once [15].
The training is done by optimizing an appropriate cost function, using a gradient-based approach. For autoencoders, a typical loss function is the mean-squared error of the reconstruction, as it can be seen in equiation 4

$$L(\theta) = \frac{1}{2N} \sum_{i=1}^{N} (x_i - y_i(\theta))^2 \tag{4}$$

where N is the number of training examples, x_i is the i-th input und y_i is the i-th reconstruction. According to the backpropagation algorithm, this loss function is then used to calculate the error for each neuron and the weights are then updated, using gradient descent.

3 Implementation

Directly using the full image, provided by the process camera, is not feasible due to the fact that it provides images in the QCIF resolution (177×144) and therefore a single image consists of 25344 pixels. As the computational time of an ANN is dependent on the number of neurons, this would lead to an unreasonable high run time for the algorithm. We therefore use only a region of interest of the size 105×105, which is then subsampled to reduce to the final size of 32×32. Figure 2 shows the preprocessing for two in-process images.

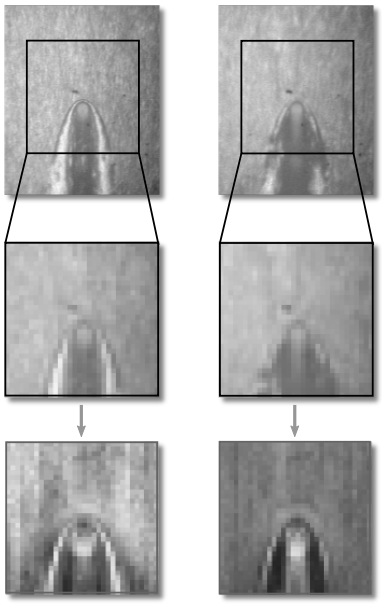

Fig. 2. Image preprocessing and reconstructed Image.The first steps shows the region of interest and the new, interpolated image. The third row shows the image, reconstructed by the autoencoder.

3.1 Autoencoder

As described in section 2.3, the autoencoder consists of two parts, the encoder and the decoder. A schematic illustration can be seen in figure 3. The first layer consists of 1024 neurons, which act as input neurons. Each neuron takes the value of its corresponding value in the image and passes it into the autoencoder. The second layer is made of 2048 neurons. To enlarge the first hidden layer is a common approach to allow the network to create a more general representation of the input. The following hidden layers decrease in their number of layers consistently until the bottleneck is reached. In the bottleneck, there are only 16 neurons. Therefore, all the information is now encoded in these 16 features. From here, the architecture is mirrored to be able to reconstruct the output, corresponding to equation 3. In total, the autoencoder consists of 11 layers. As the autoencoder is used as a feature extractor, the decoder is only used during the training. Once the autoencoder is learned, only the encoder up to the bottleneck is implemented.

Training an ANN involves the choice of many hyper parameters, such as learning rates, momentum etc. for improving the backpropagation algorithm. These parameter have a huge influence, not only on the learning time, but also on the final performance. We have therefore applied different techniques and

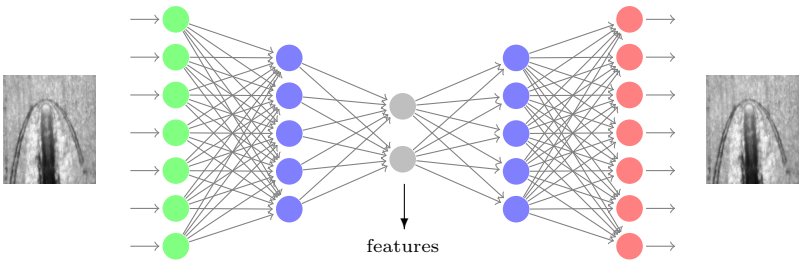

Fig. 3. Schematic illustration of training an autoencoder. The image is fed into the network on the left side, compressed into the feature space and then reconstructed. The weights are adjusted such that the output matches input.

approaches, but going into detail is beyond the scope of this paper, so we refer to [17] for further information.

3.2 Parallelization and Cache-Efficiency

As no shortcut or lateral connections are built in the network, the output computation of each neuron can be performed layer-wise and independent of the other neurons of the corresponding layer. Each layer can be computed by two nested for loops of where the outer loop which iterates over the current layer, can be parallelized as illustrated by figure 4.

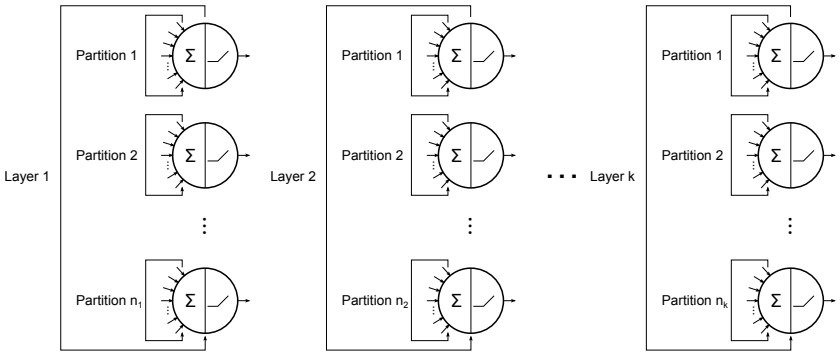

Fig. 4. Partitioning the network calculation into multiple sub tasks.

As each outer loop cycle has to compute an inner loop of a few hundred to thousand iterations, parallelization of this task seems reasonable. Indeed, the parallel computation resulted in a 3.5 times faster execution of the network on an Intel Quad-Core i7 with 3.2 GHz CPU speed, compared to the serial computation. On this platform, the calculation took about 8 to 9 milliseconds

after implementing the parallel structure, while it took about 29-30 milliseconds with the serial solution.

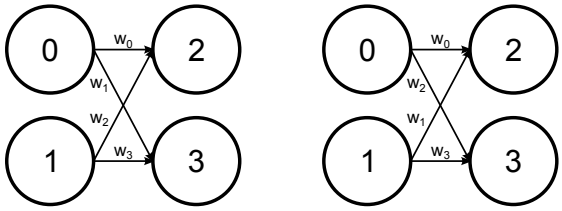

Fig. 5. Cache inefficient (left) vs. cache efficient sorting (right) by reordering weight w_1 and w_2

However, 8 milliseconds is still too slow to perform real-time capable image processing. Therefore, the weights were re-arranged in memory such that they could be accessed in a cache efficient way. The weights should be stored in a sequence container (like `std::vector`) and be iterated over in steps by 1. This can be achieved by sorting the weights based on their input neuron, not on their connection. Figure 5 shows a comparison of these two sort methods. After changing the sort method from method 1 to method 2, the performance of the network increased by a factor of 5, resulting in a total execution time of 1.6 milliseconds for one complete network computation.

4 Experiments

The experiments were conducted on industrial laser welding. The material is zinc-coated steel and the configuration is a classical overlap weld. The dataset is divided into two processes, where the used sensor hardware is slightly different. For the first setup, a collimation lens of $150mm$ and a focusing lens of $300mm$ were used, which results in a zoom factor of $3:2$ - this dataset will be referred to as process 1. In the second setup, the collimation lens was $125mm$ and the focusing lens was $250mm$, which results in a zoom factor of $2.5:2$. The images, taken with this setup will be referred to as process 2. Therefore the image, taken by the camera is slightly different, which has an influence on the data processing. To compare the two algorithms for dimensional reduction, the following approach was taken: The measure for the performance of both algorithms was the reconstruction error, which indicates how well the algorithm is able to reconstruct the original image from the new representation. Both algorithms transformed the original data into a new representation of 16 features. First, the two datasets were learned and reconstructed independently, in a second step both algorithms learned both datasets. The reconstruction errors for both approaches can be seen in table 1.

Table 1. Mean reconstruction error

Algorithm	Trained on Process	Tested on Process	Mean reconstruction error
PCA	1	1	0.006
	2	2	0.005
	1 + 2	1	0.002
	1 + 2	2	0.004
DNN	1	1	0.024
	2	2	0.011
	1 + 2	1	0.012
	1 + 2	2	0.014

The results indicate that the PCA performs better in terms of reconstructing the original image. However, as the purpose of the dimensional reduction is not the reconstruction, this error measure alone is not sufficient to evaluate the performance of both algorithms. We therefore trained a support vector machine (SVM), using a MLP-kernel, which functions according to equation 1, to classify the images, regarding the quality of the corresponding welding seam. This measure is far more informative, as it compares the meaningfulness of the features. The results can be seen in table 2.

Table 2. Mean classification error

Algorithm	Trained on Process	Tested on Process	Mean classification error
PCA	1	2	0.9 ± 0.003
	2	1	0.57 ± 0.08
DNN	1	2	0.59 ± 0.14
	2	1	0.67 ± 0.04

For both algorithms, the representation was learned over both processes, then the SVM was learned for one process, using this representation and applied to the other process. The results show that if the SVM is learned on the process 1 and applied on process 2, the neural network representation is performing significantly better. For the reverse case, both SVMs perform almost equally—however it has to be noted that learning the SVM on process 2 did fail to converge, if the complete dataset has been used. Therefore, the outcome of this particular experiment has limited significance.

5 Discussion

In this paper the usage of a deep autoencoder for real time feature extraction in laser welding has been shown. By making use of parallelization, a network with in total 4884 neurons and $4,858,640$ connections has been shown to process the inputs within 1.6 ms. The reconstruction error of the autoencoder has been

compared to the performance of classical approach, namely the PCA. It has been shown, that while the PCA has a lower reconstruction error, the autoencoder outperforms the classical approach if the features are later used for classification, which indicates the features from the neural network contain more information. As small deviations in the inputs are a common problem in laser welding, due to the setup procedure and different hardware setups, the autoencoder can be considered superior for feature extraction. It further has been shown, that even a neural network with thousands of neurons can be implemented to perform within our real time requirements of 2 ms.

6 Conclusion

This work provides knowledge on how to use neural networks for image processing in laser welding. As the neural network is able to be trained unsupervised and has shown its capability of extracting meaningful features in a robust and fast way, it promises to address key issues in modern industrial production and image processing. We described the architecture of the deep autoencoder and the implementation details to ensure the computation to be fast enough. To our knowledge, this is the first demonstration of a neural network, consisting of several thousand neurons, to be used for feature extraction in laser welding within real time constraints. However, as robust representation is a key attribute for intelligent systems, we do not want to limit this approach to laser welding. The results indicate that this approach is transferable to other industrial processes, using camera based process observation.

Acknowledgments

The project CCLW is performed within the framework of the European funding program Eurostars and is funded by the Federal Ministry of Education and Research. We also would like to thank all project partners, namely the Precitec GmbH & Co. KG, Germany and IREPA LASER, Illkirch, France, who provided the data. The authors would also like to thank Elias Reichensdörfer for many interesting discussions his assistance on the parallelization.

References

1. P. Zipkin, “The limits of mass customization.” *Harvard Business Review*, vol. 75, pp. 91-101, 1997.
2. J. Shao and Y. Yan, “Automated inspection of micro laser spot weld quality using optical sensing and neural network techniques,” *Proceedings of the IEEE Instrumentation and Measurement Technology Conference*, pp. 606–610, 2006.
3. I. Wersborg, K. Schorp, T. Bautze, and K. Diepold, “Multiple sensors and artificial neural networks in a cognitive technical system for laser welding,” *2009 5th International Conference on Intelligent Sensors, Sensor Networks and Information Processing* , pp. 109–114, 2009.

4. A. I. Gavrilov, "Development of automatic control system for laser welding using neural network technology," *International Conference on Control*, vol. 1, pp.278–283, 1998.
5. L. Nicolosi, R. Tetzlaff, F. Abt, A. Blug, and H. Hofler, "Cellular neural network (CNN) based control algorithms for omnidirectional laser welding processes: Experimental results," *2010 12th International Workshop on Cellular Nanoscale Networks and Their Applications* , pp. 1–6, 2010.
6. Y. Bengio, A. Courville, P. Vincent, "Representation Learning: A Review and New Perspectives," *Pattern Analysis and Machine Learning*, vol. 35, no. 8, p. 1798–1828, 2013.
7. P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, p. 33713408, 2010.
8. L. J. P. v. d. Maaten, E. O. Postma, and H. J. v. d. Herik, *Dimensionality Reduction: A Comparative Review*, 2008.
9. Z. J. Daruwalla, P. Courtis, C. Fitzpatrick, D. Fitzpatrick, and H. Mullett, "An application of principal component analysis to the clavicle and clavicle fixation devices," *Journal of Orthopaedic Surgery and Research*, vol. 5, no. 1, p. 21 – 28, 2010.
10. I. Gergen and M. Harmanescu, "Application of principal component analysis in the pollution assessment with heavy metals of vegetable food chain in the old mining areas," *Chemistry Central Journal*, vol. 6, no. 1, p. 156 – 168, 2012.
11. K. I. Kim, K. Jung, and H. J. Kim, "Face recognition using kernel principal component analysis," *IEEE Signal Processing Letters*, vol. 9, no. 2, pp. 40–42, 2002.
12. M. Jager, S. Humbert, and F. Hamprecht, "Sputter tracking for the automatic monitoring of industrial laser-welding processes," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 5, pp. 2177–2184, 2008.
13. F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.
14. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.
15. Y. Bengio, "Learning deep architectures for ai," *Journal Foundations and Trends*, vol. 2, no. 1, pp.1–127, 2009.
16. G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.
17. Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *Lecture Notes in Computer Science*, vol. 7700, 2012.