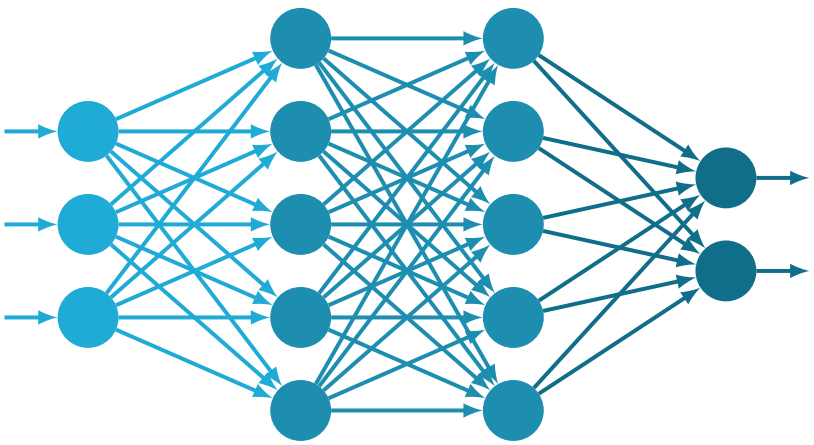
KU LEUVEN

# Artificial Neural Networks & Deep Learning

## Exercise Session 3

**Prof. Dr. ir. Johan A. K. Suykens**

**Assistants**
**Sonny Achten**
**Bram De Cooman**
**David Winant**
**Xinjie Zeng**



2023-2024

SESSION 3

# Deep Feature Learning

**Note:** There is no need to repeat the information provided during the guided session. Answer the questions as concise and accurate as possible. Use graphical and/or tabular presentation. The *page limit* for this exercise is *4 printed pages*, including figures.

This assignment investigates different deep learning models, including Autoencoders, Stacked Autoencoders, Convolutional Neural Networks and Transformers.

## 3.1 AUTOENCODERS AND STACKED AUTOENCODERS

In this section, we consider two neural network architectures: autoencoders and stacked autoeconders. Since the latter consists of the former as basic modules, we first introduce the autoencoders.

### Autoencoders

*This is an introductory section. You will not have to include this part in your report.*

An autoencoder neural network is a self-supervised learning algorithm where the target vectors are the same as the input vectors, i.e., $\mathbf{y}_i = \mathbf{x}_i, \forall i = 1, \ldots, N$. Figure 3.1 shows an example of an autoencoder. Specifically, the autoencoder tries to learn a parametric function such that $\hat{\mathbf{x}} := h_{\mathbf{W},\mathbf{b}}(\mathbf{x}) \approx \mathbf{x}$ where the parameters are updated using backpropagation. In other words, it learns an approximation to the identity function, so as to have an output $\hat{\mathbf{x}}$ that is similar to the input $\mathbf{x}$. The identity function seems a particularly trivial function to learn, however, by placing constraints on the network, such as 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 performing a non-linear dimensionality reduction, thereby overcoming certain limitations of dimensionality reduction algorithms such as linear PCA. The network can be trained by minimizing the appropriate reconstruction error.

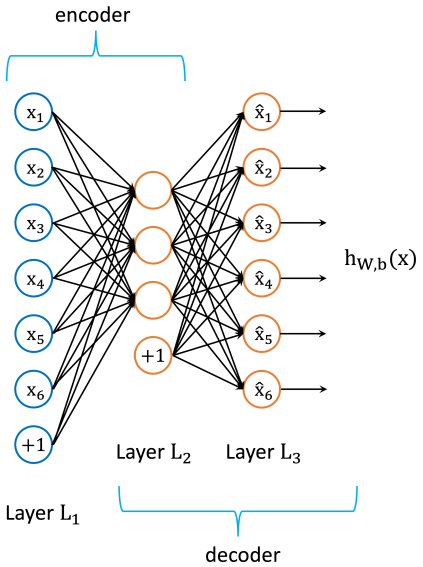



Figure 3.1: An example of Autoencoder. The part from the input layer to the hidden layer is called the encoder and the part from hidden layer to the output layer is called the decoder.

### Exercises

*This is an introductory exercise. You will not have to include this in your report.*
Please refer to the file `ae_sae.ipynb` for detailed experiments. Run the autoencoder experiments in the file and answer the following questions:

1. Conduct image reconstruction on synthetic handwritten digits dataset (MNIST) using an autoencoder. Note that you can tune the number of neurons in the hidden layer (`encoding_dim`) of the autoencoder and the number of training epochs (`n_epochs`) so as to obtain good reconstruction results. Can you improve the performance of the given model?

### Stacked Autoencoders

*Provide your answers to the questions in this section in your report.*
A stacked autoencoder is a neural network consisting of multiple layers of sparse autoencoders in which the outputs of each layer become the inputs of the successive layers. 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 3.2.

A good way to obtain parameters for a stacked autoencoder is to use *greedy layer-wise training*. We first train the 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 updating the parameters of all layers at the same time.

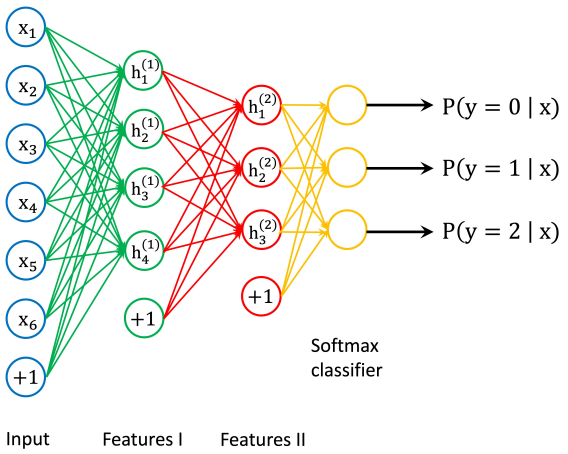



Figure 3.2: An example of a stacked autoencoder used for classification.

### Exercises

Please refer to the file `ae_sae.ipynb` for detailed experiments. Please run the Stacked Autoencoder experiments in the file and answer the following questions:

1. Conduct image classification on MNIST using an stacked autoencoder. Are you able to obtain a better result by changing the size of the network architecture? What are the results before and after fine-tuning? What is the benefit of pretraining the network layer by layer?

## 3.2 CONVOLUTIONAL NEURAL NETWORKS

*Provide your answers to the questions in this section in your report.* In this section, we first introduce the convolutional neural networks (CNNs) which is a category of neural networks consisting of convolutional layers.

## Convolutional Neural Networks

Convolutional Neural Networks (CNNs) is a type of neural network architecture which uses the concept of local connectivity. Different from the fully connected layers where all nodes from subsequent layers are connected, in convolutional layer, neurons that are close to each other are likely to be more related than neurons that are further away. For example, in images where the data points represent pixels. Pixels that are close are likely to represent the same part of the image, while pixels that are far away can represent different parts. Based on this idea, it is reasonable to assume that the parameters learned at some local position of the image would be useful at other location.

**Convolution Arithmetic: A Toy Example** Consider an input vector $\mathbf{x} = \boxed{2}\boxed{5}\boxed{4}\boxed{1}\boxed{3}\boxed{7}$, which we want to convolve with a 1D kernel $\mathbf{k} = \boxed{1}\boxed{0}\boxed{1}$. The kernel slides across the input vector at each step the product between each element of the kernel and the input element it overlaps is computed and the results are summed up to obtain the output. This step-size is formally called as *stride*. With no-padding and unit stride, this will produce the output $\mathbf{y} = \boxed{6}\boxed{6}\boxed{7}\boxed{8}$. For extensions to 2D cases with various strides and padding, please refer to MILA's guide [1].

## Exercises

Please refer to the file `cnn.ipynb` for detailed experiments. For the second question, please run the CNNs experiments in the file:

1. Answer the following questions:

   - Consider the following 2D input matrix.

   $$\mathbf{X} = \begin{bmatrix} 2 & 5 & 4 & 1 \\ 3 & 1 & 2 & 0 \\ 4 & 5 & 7 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix},$$

   Calculate the output of a convolution with the following 2x2 kernel with no padding and a stride of 2.

   $$\mathbf{K} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

   - How do you in general determine the dimensionality of the output of a convolutional layer?
   - What benefits do CNNs have over regular fully connected networks?

2. The file `cnn.ipynb` runs a small CNN on the handwritten digits dataset (MNIST). Use this script to investigate some CNN architectures. Try out different amounts of layers, combinations of different kinds of layers, number of filters and kernel sizes. Note that emphasis is not on experimenting with batch size or epochs, but on parameters specific to CNNs. Pay close attention when adjusting the parameters for a convolutional layer as the dimensions of the input and output between layers must align. Discuss your results. Please remember that some architectures will take a long time to train.

## 3.3 SELF-ATTENTION AND TRANSFORMERS

*Provide your answers to the questions in this section in your report.* In this section, we work on the self-attention mechanism, which is the core component in Transformers.

### Self-attention Mechanism

For many NLP and visual tasks which we train our deep models on, features appear on the input text or visual data often contributes unevenly to the output task. For instance, in a translation task, not the entirety of the input sentence will be useful for the model to generate a certain output word. As in Figure 3.3, we can explain the relationship between words in one sentence or close context. When we see "eating", we expect to encounter a food word very soon. The color term describes the food, but probably not so much with "eating" directly. Hence, attention mechanisms are developed to improve the network's capability of orienting perception onto parts of the data, and to allow random access to the memory of processing previous inputs.

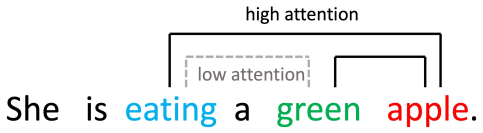



Figure 3.3: One word "attends" to other words in the same sentence differently.

Self-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence. It has been shown to be very useful in machine reading, abstractive summarization, or image description generation. Scaled Dot-Product attention is a self-attention mechanism introduced in the Transformer architecture [3]: Let $\{\mathbf{x}_i\}_{i=1}^N$ be the input data sequence. In self-attention, the queries, keys and values output the linear projections of the input sequence:

$$q(\mathbf{x}_i) = W_q\mathbf{x}_i, \quad k(\mathbf{x}_i) = W_k\mathbf{x}_i, \quad v(\mathbf{x}_i) = W_v\mathbf{x}_i.$$

The canonical self-attention is with "softmax" activation applied for bringing non-linearity and positive outcomes, yielding the attention scores:

$$\mathbf{A}_{i,j} = \text{softmax}\left(\langle W_q\mathbf{x}_i, W_k\mathbf{x}_j\rangle / \sqrt{d_k}\right), \quad i,j = 1,\ldots,N,$$

where $\sqrt{d_k}$ is a scaling and $d_k$ is the dimension of key. The attention output is obtained by $\mathbf{o}_i = \sum_{j=1}^N v(\mathbf{x}_j)\mathbf{A}_{i,j}$, $i = 1,\ldots,N$.

### Exercises

Please refer to the file `attention.ipynb` for detailed experiments. Please run the self-attention mechanism experiments in the file and answer the following questions:

1. Please run both the NumPy and PyTorch implementations of the self-attention mechanism. Can you explain briefly how the dimensions between the queries, keys and values, attention scores and attention outputs are related? What do the query, key and value vectors represent? Note that the attention mechanism will also be discussed in lecture 11.

### Transformers

"Attention is All you Need" [3], without doubt, is one of the most impactful and interesting papers in 2017. It presents a lot of improvements to the soft attention and make it possible to do seq2seq modeling without recurrent network units. The proposed "transformer" model, given in Figure 3.4, is entirely built on the self-attention mechanisms without using sequence-aligned recurrent architecture. While we described the self-attention variant of a single attention head above, we can extend it to multi-head self-attention. Moreover, the transformer network concatenates multiple sets of these weights as multiple heads in the attention mechanism. This forms the most important building block of the Transformer Architecture. For a more detailed discussion of the Transformer Network, please refer to the original paper [3].

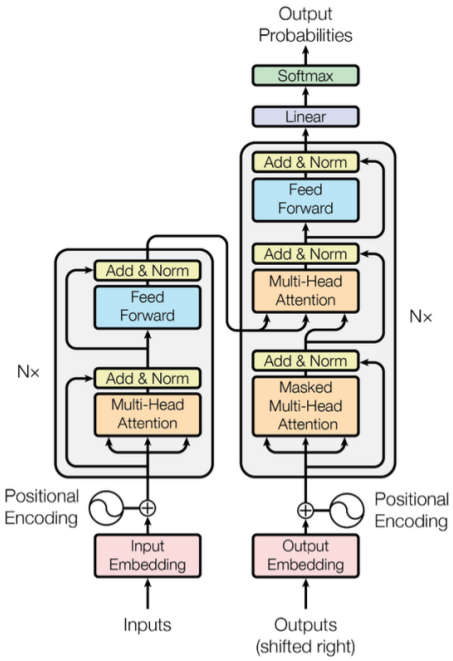



Figure 3.4: Full Transformer Architecture.

### Exercises

Please refer to the file `attention.ipynb` for detailed experiments. Please run the Vision Transformer (specifically designed for computer vision tasks) experiments in the file and answer the following questions:

1. Please train the Transformer on the MNIST dataset. You can try to change the architecture by tuning `dim`, `depth`, `heads`, `mlp_dim` for better results. You can try to increase or decrease the network size and see whether it will influence the prediction results much. Note that ViT can easily overfit on small datasets due to its large capacity. Discuss your results under different architecture sizes.

## 3.4 REFERENCES

[1]. Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016).

[2]. He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778. 2016.

[3]. Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." Advances in neural information processing systems 30 (2017).