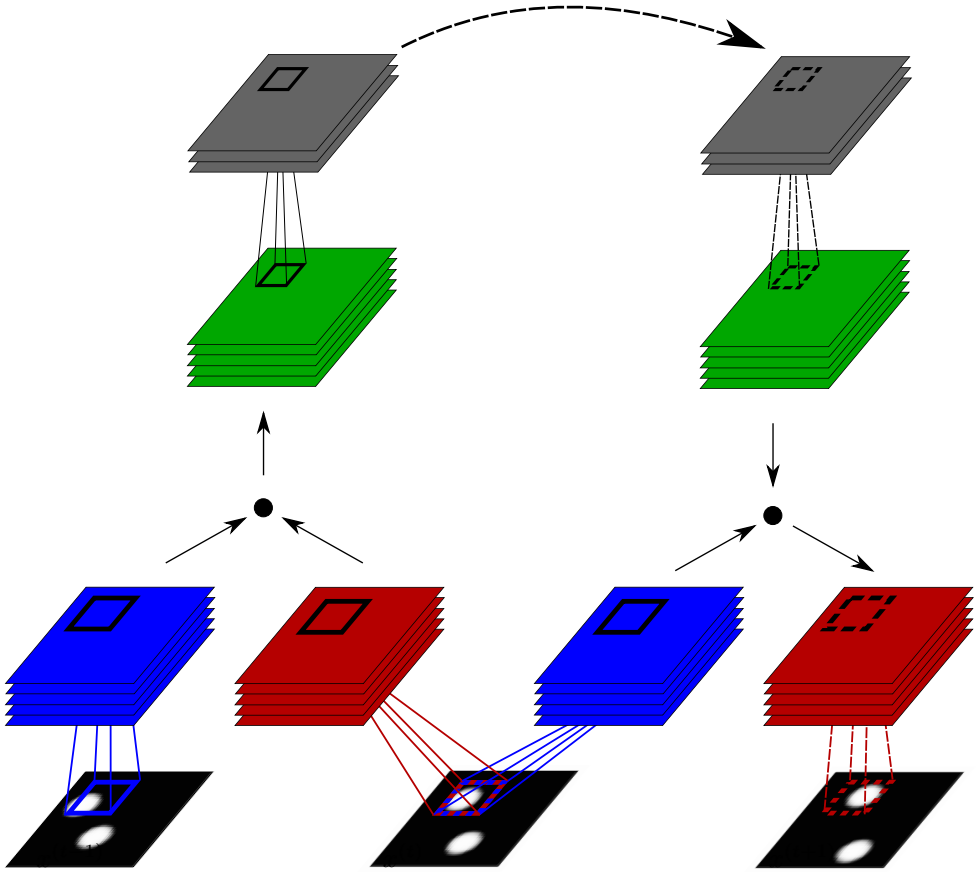
CHALMERS



Modeling spatiotemporal information with convolutional gated networks

Master's thesis in Applied Mechanics

FILIP DE ROOS

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2016

MASTER'S THESIS IN APPLIED MECHANICS

Modeling spatiotemporal information with convolutional gated networks

FILIP DE ROOS

Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2016

Modeling spatiotemporal information with convolutional gated networks
FILIP DE ROOS

Master's thesis 2016:83
ISSN 1652-8557
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
Graphical depiction of the convolutional factored gated autoencoder developed in this thesis to improve the predictive gating pyramid. The blue, red and green stacks are convolutional feature maps detecting spatiotemporal synchrony. The grey stack encodes local transformations between two inputs, which can be used to predict the next input. Full lines leading into a stack represent normal convolutions and dashed lines represent transposed convolutions.

Chalmers Reproservice
Göteborg, Sweden 2016

Modeling spatiotemporal information with convolutional gated networks
Master's thesis in Applied Mechanics
FILIP DE ROOS
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

In this thesis, a recently proposed bilinear model for predicting spatiotemporal data has been implemented and extended. The model was trained in an unsupervised manner and uses spatiotemporal synchrony to encode transformations between inputs of a sequence up to a time t, in order to predict the next input at $t + 1$. A convolutional version of the model was developed in order to reduce the number of parameters and improve the predictive capabilities. The original and the convolutional models were tested and compared on a dataset containing videos of bouncing balls and both versions are able to predict the motion of the balls. The developed convolutional version halved the 4-step prediction loss while reducing the number of parameters by a factor of 159 compared to the original model. Some important differences between the models are discussed in the thesis and suggestions for further improvements of the convolutional model are identified and presented.

Keywords: spatiotemporal, predictive gating pyramid, convolutional neural network, unsupervised learning, bouncing balls.

Acknowledgements

Firstly, I would like to thank my supervisor Jörg Wagner for making the thesis possible, continuously supporting me and providing interesting discussions. I would also like to thank the CR/AEY2 group at Robert Bosch GmbH for allowing me into the company and in particular Volker Fischer for providing valuable insights into the field of deep learning.

A special thanks to my examiner Mattias Wahde for his meticulous proofreading of my thesis and helpful advice.

Furthermore, I would like to extend a thanks to the other students in the CR/AEY2 for their friendship and extensive Tischkicker tutoring. Lastly I would like to thank Maren for her unwavering support and proofreading throughout the thesis.

Contents

1 Introduction

Autonomous vehicles have gained a lot of interest in recent years. Having a system control the behaviour of the vehicle can lead to multiple advantages, such as fewer accidents, improved fuel efficiency and more efficient parking [31]. Due to recent success for deep learning models to reach state-of-the-art performance in multiple categories of machine learning, deep learning models provide valuable functionality to autonomous vehicles. The environment where vehicles operate is dynamic and continuously changing. In order to navigate these environments safely, the temporal and spatial information provided must be processed to predict object or pedestrian movement and assess dangerous situations.

1.1 Problem formulation

For a model to predict the movement of an object, it requires both spatial and temporal information which corresponds to where and how the object moves. A module that predicts possible trajectories of objects can improve the performance of tracking and classification modules. By providing prior knowledge of the next frame in the case of a video, it can help other modules to de-noise the input, resolve ambiguities, and direct focus for tracking [51]. The problem approached in this thesis consists of predicting the continuation of a sequence with spatial and temporal information.

1.2 Related work

The field of deep learning has produced models that achieve state-of-the-art performance on multiple tasks relating to computer vision, such as image classification [14, 28] or image segmentation [59]. In the case of classification, a network is supposed to determine which classes of objects are present in a picture. For segmentation, a network is supposed to assign class labels to every pixel in the presented image. Both tasks are applied to single images and use the spatial information present in the pictures. The currently most popular models within deep learning that operate on spatial information use convolutional layers to form *convolutional neural networks* (CNNs), which will be explained in Chapter 2.

Temporal information requires sequential data which is inherently found in videos, speech or other data from time series. Due to the natural occurrence of such data, there is plenty of data to analyse with widespread applications such as speech recognition and natural language processing [2, 29], action recognition [23] or weather prediction [58]. CNNs can learn to model spatiotemporal data by extending the normal 2D convolution to also convolve over time [21, 23, 55]. Consecutive frames are stacked to form a video volume and a 3D convolution is performed across the volume, treating the temporal domain as another spatial dimension. Such models have successfully been applied to classification tasks like action recognition, but not for prediction.

Efficiently combining spatial and temporal information in deep learning, models have proven to be difficult and many approaches have been tested. The most common approach is to use networks with recurrent connections, *i.e. recurrent neural networks* (RNNs) [9, 13]. Such networks contain a hidden state h_t that is sequentially updated and depends on the current input x_t and the previous value of the hidden state h_{t-1} [29]. RNNs behave as dynamical systems [18] and focus mainly on the temporal part of sequences, which leads to a manifold of a specific system being learnt [2]. For some applications, it can be enough to know the possible evolutions of a specific system, in other cases it might require more generalisation.

Gregor *et al.* [12] used RNNs with an attention mechanism to update parts of an image in a sequential manner, which can be seen as a form of prediction where the current update of the image depends on the previous. Building upon the work of Gregor [12], Kahou *et al.* [22] applied a recurrent attention mechanism to track single objects in a video sequence. The model makes use of the spatiotemporal information and learns to expect certain evolutions to accurately track objects but it does not generate a prediction of the input. Similarly, Shah *et al.* [46] used a RNN to predict the outcome of three-point shots in basketball as successful or a miss. The model is closer to a classifier than a sequence generator in that regard. The model uses ball trajectories and accurate predictions of the trajectories could improve such a classifier.

RNNs are known to suffer from the vanishing or exploding gradient problem, often rendering them incapable of learning [16]. The *long short-term memory* (LSTM) [17] is a popular generalisation of the RNN with gated connections and a cell-state which alleviates the vanishing gradient problem. LSTMs can be combined in an encoder-decoder structure to learn temporal transformations and predict further ahead [48, 8]. One stack of

LSTMs is used to encode a sequence and two to decode it. One stack is used to reconstruct the input and the second to predict the continued sequence. Such a model was able to predict trajectories but obtained a strong bias for the training set. The model proposed in [48] was trained on predicting the trajectories of two objects. When provided sequences of one or three objects, it would try to change the data to two objects by "imagining" another object or merge objects.

The most prominent models to use spatiotemporal information combine the spatial properties of CNNs and the temporal of RNNs [6, 43, 44]. By replacing the normal matrix-vector products found in RNNs with convolutions from CNNs, the model proposed by Ranzato *et al.* [44] could predict and generate a few frames ahead. When training on the *mean-square error* (MSE) of the predicted image, the predictions would be a blurred version of the last frame [34]. To circumvent the averaging, the authors instead classified patches of images using k-means clustering and stored them in a dictionary. The goal was then to predict the identity of the next patch in the dictionary. The quality of the predictions improved but the networks could only predict a few frames ahead before freezing all movement. Although using a dictionary of patches improved the resulting predictions, it also introduced an arbitrary dictionary size which determined how expressive a prediction could be. Finding appropriate training criteria for image prediction have proven nearly as difficult as the problem of predicting the images [48]. Despite other criteria having been suggested [44, 34, 42], MSE is still a a popular choice.

Stacking of convolutional recurrent layers similar to Ranzato *et al.* [44] with additional connections between the layers yielded good results for object-class segmentation [43]. A side-effect of the stacking was a good ability to track moving objects as well as inferring motion between frames but not to predict ahead. The model proposed by Ondruska *et al.* [41] modified a convolutional RNN by replacing the hidden state with a belief tracker. The belief tracker consisted of a spatial map of the input showing where the network believed an object to be, based on visual input. By training on what the model will see in a future frame (not what is present), the network had to learn the most probable placement of objects in the belief tracker. The network learned to handle partial occlusion and track objects despite full occlusion.

Convolutional operations have also been incorporated into LSTMs for spatiotemporal modelling [42, 58] similar to the encoder-decoder structure proposed by Srivastava *et al.* [48]. Due to the long-range temporal dependencies offered by the LSTM and the spatial computations of convolutions, these models consistently outperformed the original fully-connected versions for modelling spatiotemporal structure. A different approach to capture spatiotemporal structure is to use bilinear models [36, 38, 39, 47, 51]. Such models learn relations between inputs which can be used for stereo vision [26], sequence prediction [40] and action recognition [53]. Compared to recurrent networks with a hidden state, the bilinear models are more transparent and easier to analyse. Most work relating to bilinear models focus on learning global transformations between inputs, *i.e.* images have been flattened into vectors to accommodate matrix-vector products. By flattening the input, a model ignores the locally correlated information present in pictures [42, 53]. Taylor *et al.* [53] modified a bilinear model to use convolutions instead of matrix-vector multiplications to use spatial data more efficiently. The model performed well on the task of action recognition in videos.

1.3 Aim

Popular approaches within the field of deep learning for input prediction rely on a hidden state that is continuously updated with information in order to sample predictions. The dynamics of such models are seldom easy to interpret making it difficult to analyse how they operate. A recently proposed bilinear model [40] learns transformations between inputs and does not depend on an explicit hidden state to model the transformations. The model provides a more transparent view into the spatiotemporal dynamics, making it possible to analyse the computations taking place.

The goal of this thesis is to improve the spatial abilities of said model by using a convolutional architecture similar to Taylor *et al.* [53] in order to model local transformations. The final model would use a hierarchical architecture to model transformations that are spatially local and that can be trained by gradient-descent. The new approach would lead to more general transformations, fewer parameters allowing for faster training and improved local predictions.

1.4 Scope and limitations

The goal of this thesis was to implement and extend a known architecture from a fully-connected setting to a convolutional one, in order to improve the spatiotemporal predictive capabilities of the model. The version was analysed on a one-dimensional data set of "1"s moving around and evaluated on a data set of video sequences of bouncing balls. The goal was to test and understand the model rather than finding the best possible parameters. Therefore, testing of functional-specific parameters like different nonlinearities and loss-functions took place but no explicit hyperparameter optimisation.

1.5 Outline

The remainder of the thesis is structured as follows. Chapter 2 presents the theory used within the thesis and consists of two parts. The first part treats the general background of artificial neural networks and how they are trained, while the second part explains the concept of spatiotemporal synchrony and the theory behind the predictive gating pyramid [40]. Chapter 3 presents the datasets and the implementation used for training and testing. The obtained results are presented and discussed in Chapter 4, comparing the original model to the convolutional. The thesis is concluded in Chapter 5, summarising the work and suggesting possible areas for future work.

2 Theory

This chapter aims to present the theory used throughout the thesis. The first part treats the basics of artificial neural networks. It explains how they are constructed in a fully-connected as well as a convolutional setting and how training is performed and can be improved. The second part focuses on the theory behind the predictive gating pyramid and presents the extension.

2.1 Artificial neural networks

There are many tasks in our everyday life that come natural to humans but have proven difficult for computers to solve, such as speech or face recognition. One way to tackle these problems is to mimic systems found in nature that can solve these tasks. An example of such a system is the human brain which through evolution has developed extraordinary abilities to process data in the form of external stimuli. The brain consists of an intricate network of neurons where stimuli are chemically transmitted in the form of neurotransmitters, from one neuron to another through the so called synapses. Depending on how often certain activity between neurons occur, some connections between neurons become stronger or weaker [15], allowing the system to adapt to and process stimuli in an optimal way. The ability to process data in a complex manner as the brain are goals for many computer-related problems. A popular approach to these problems is to construct artificial neural networks that are inspired by their biological counterpart and trained to solve complex problems. In this section, the basics of artificial neural networks and how they are trained will be explained.

2.1.1 Multilayer perceptron

The basic building block of artificial neural networks are the artificial neurons, sometimes referred to as perceptrons. A neuron is provided a set of inputs x_i that each are weighted and summed over. The result is passed through a nonlinearity to obtain the activation of the neuron [20, 13]. It is common to add a bias term b_j to the activation of a neuron. The activation can be seen in Eq. (2.1) with $\phi(\cdot)$ representing the nonlinear activation function.

$$h_j = \phi\left(\sum_i w_{ji} x_i + b_j\right) \tag{2.1}$$

The neurons are combined into layers known as hidden layers and stacking multiple layers forms a network. These networks are commonly referred to as *multilayer perceptrons* (MLPs). Figure 2.1 shows an MLP with four input parameters, two hidden layers with six and five artificial neurons respectively and an output layer of dimension three. Each neuron in the MLP takes the activation of every neuron in the previous layer as input and the layers are therefore referred to as *fully connected* (FC).

2.1.2 Loss function and training

Training of artificial neural networks can be achieved in many different ways depending on the architecture. The most common is to use a known objective that the output of the network should reproduce. The performance of the network is determined by how well the output corresponds to the provided objective given a metric known as the loss function. The loss function defines the error between the output of the network and the objective. The goal of training a network is to minimise the error. A method known as backpropagation of errors developed by Werbos [56] and later popularised by Rumelhart [45] uses the chain rule to iteratively compute the gradients for each layer in the network with respect to the loss function. The errors from subsequent layers in the network are propagated backwards to calculate the gradient for the weights of a specific layer. Equation (2.2) shows how the loss ($\mathcal{L}$) between the output of the network (o) and the target (t) are propagated backwards to get the gradient with respect to weight W in layer (l).

$$\frac{d\mathcal{L}(o,t)}{dW^{(l)}} = \underbrace{\frac{\partial \mathcal{L}(o,t)}{\partial o}\frac{\partial o}{\partial h^{(N-1)}}\frac{\partial h^{(N-1)}}{\partial h^{(N-2)}}\cdots}_{\text{backpropagated error}}\frac{\partial h^{(l+1)}}{\partial W^{(l)}} \tag{2.2}$$

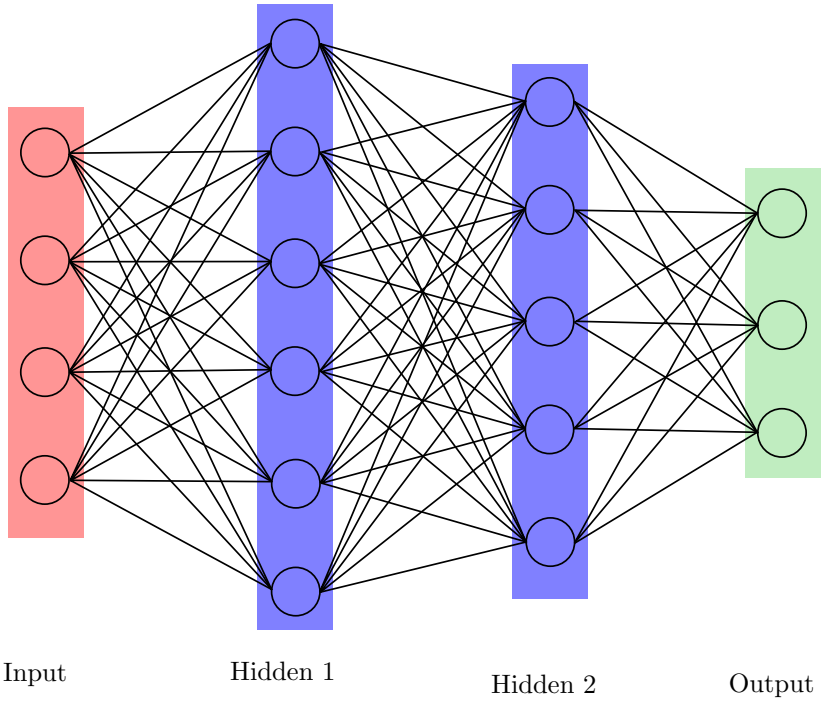


Figure 2.1: *A three layer MLP with four variables as input and three as output. The input layer is followed by two hidden layers with six and five hidden neurons. The network calculates an output of dimension three which is the number of artificial neurons in the output layer.*

For the chain rule to be applicable, it is necessary that the nonlinear activation function ϕ in Eq.(2.1) is differentiable [13].

Once the gradient with respect to a parameter is obtained, the parameter is updated to minimise the loss function by changing the value according to a chosen gradient-based optimisation method. The most common optimisers are based on the *stochastic gradient descent* (SGD) which updates the parameter value by changing the value in the opposite direction of the gradient. Equation (2.3) shows the updating procedure of the SGD with the gradient defined as Eq. (2.2) and η being the learning rate modulating the magnitude of the update.

$$W^{(l)} = W^{(l)} - \eta \frac{d\mathcal{L}(o,t)}{dW^{(l)}} \tag{2.3}$$

The datasets used for training a network can be large and smaller batches of the set called *mini-batches*. It is assumed that the gradient over a mini-batch approximates the gradient over the entire training dataset [4]. The loss function is then defined by the sum of all individual losses in the batch, see Eq. (2.4) with N being the number of samples in the batch.

$$\mathcal{L}_{\text{batch}} = \sum_{i=1}^{N} \mathcal{L}(o_i, t_i) \tag{2.4}$$

More advanced versions of gradient descent have been developed to speed up the training procedure. A simple improvement is to add momentum to the updates.

$$\begin{aligned} v_{t+1} &= \mu v_t - \eta \frac{d\mathcal{L}(o,t)}{dW^{(l)}} \\ W^{(l)}_{t+1} &= W^{(l)}_t + v_{t+1} \end{aligned} \tag{2.5}$$

The equation above shows an SGD update with momentum. The parameter v accumulates previous gradients as a velocity which decays by a factor $\mu \in [0, 1]$ for every iteration. The momentum update is further improved in the Nesterov accelerated momentum which calculates the gradient at the next iteration to perform an improved update of the weights at the current iteration [52]. A different approach is used in RMSProp which modulates the update of each parameter by maintaining an average of previous gradients [54]. One of the

currently most popular optimisers is the Adam algorithm [24], which is similar to RMSProp but contains an additional momentum term for the gradients to get smoother updates.

2.1.3 Nonlinear activation functions

The activation of a neuron is determined by the inputs, the weights, possibly a bias and an activation function, see Eq. (2.1). The activation function transforms the sum of weighted inputs in a nonlinear way so that networks can encode complex nonlinear functions of the input [20]. Stacking multiple layers increases the number of nonlinear transformations and thus also the complexity of the computational graph the network represents. The first nonlinear activation function proposed by McCulloch and Pitts was a binary thresholding operation [35]. The activation of a neuron is 1 if the input is larger than a threshold θ and 0 otherwise. If the constant threshold θ is combined with the bias b in Eq. (2.1), the binary threshold is seen in Eq. (2.6).

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases} \tag{2.6}$$

There are differentiable activation functions with similar asymptotic behaviour such as the sigmoid and tanh functions. These functions squash the input to the range $[0, 1]$ and $[-1, 1]$ respectively. If a neuron gets an activation close to either of the limits, it is said to be saturated. The saturation leads to a gradient close to zero for the neuron. If this occurs for many neurons, the network will fail to learn since the gradient effectively dies out. To avoid the saturation, it is possible to use other activation functions, *e.g.* the *rectified linear unit* (ReLU), which can result in up to six times faster training of deep networks compared to the sigmoid [28]. The ReLU thresholds values smaller than zero and lets positive values flow unhindered, see equation (2.7).

$$f(x) = \max(0, \mathrm{x}) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{2.7}$$

Due to the large speed-ups gained during training, the ReLU is a popular choice for activation function. The ReLU can also lead to sparse activations. In a randomly initialised layer, approximately 50% of the weighted sums have values larger than 0 and are able to propagate information. A downside to the sparsity of the ReLU is that the thresholding can lead to parts of the network never activating. If a neuron never activates due to negative stimulus, it will not affect the overall loss function and the weights will never be updated. Other functions have been proposed to remedy the inactivity of neurons, *e.g.* the leaky ReLU ($f(x) = \max(\alpha \mathrm{x}, \mathrm{x})$) [33] or the exponential ReLU ($f(x) = x(x > 0) + (\exp(-\mathrm{x}) - 1)(\mathrm{x} < 0)$) [5]. Using a more advanced version of the ReLU does not always result in better performance and the ReLU is therefore still used to a large extent.

2.1.4 Regularization

The goal of training an artificial neural network is to apply the network to the same or a similar task but for other data than the training data. The network must then be able to generalise its computations to previously unseen data [9]. To measure how well a network generalises to other data, it is common to monitor the loss function over both the training set and a validation set. The loss over the validation set shows how well the output of the network generalises to unseen data. If the loss over the validation set starts to increase while the loss over the training set still decreases, the network has overfitted to the training data and will not perform as well on new data. By monitoring the loss over the validation set, it is possible to stop the training when the validation loss has reached a minimum, see Fig. 2.2.

Regularization techniques have been developed to allow networks to prevent overfitting during training. One way to regularize the network is to impose a norm on the weights, such as the L_1 or the L_2 norm and add the contributions to the overall loss function.

$$\mathcal{L}_{tot}(x, x') = \mathcal{L}(x, x') + \lambda_1 ||w||_1 + \lambda_2 \frac{1}{2} ||w||_2^2 \tag{2.8}$$

The equation above shows an example of the overall loss function with λ_1 and λ_2 being hyperparameters controlling the impact of the added norm restrictions. The two norms have different effects on the weights. The L_1 norm encourages sparsity in the weights, so fewer weights contribute to the activation of a neuron, whereas the L_2 norm limits the magnitude of the weights to small values

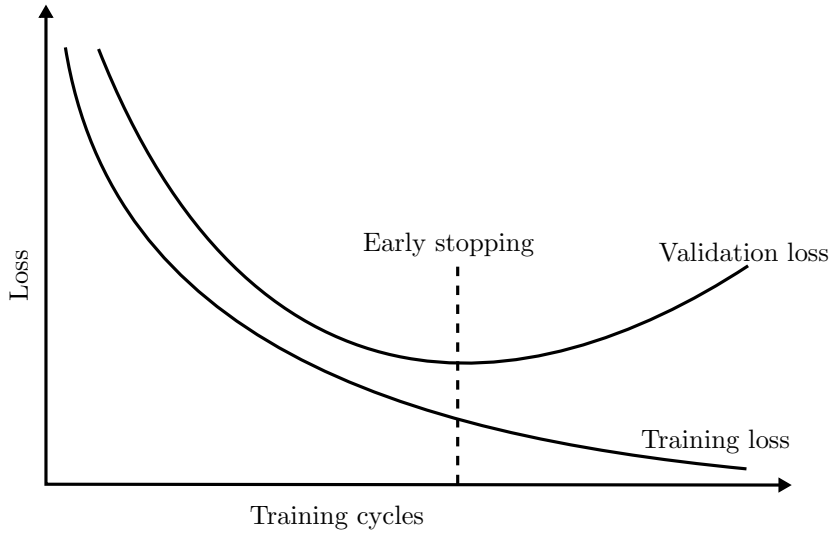


Figure 2.2: *Graph of the loss for a network overfitting to the training set. By stopping training at the early stopping mark where the loss over a validation set is minimal, the network has reached the optimal capacity to generalise the computations to data outside of the training set.*

Regularization can also be achieved by corrupting the input with the addition of a small amount of noise or by fixing a random set of subsequent activations to zero. The latter is known as dropout [49] where a percentage p of the activations are set to 0 during training. Dropout forces the network to assign relevant information for the computations to multiple activations, leading to a more robust computational graph that is less sensitive to deviations in the input. Figure 2.3 shows how activations in a MLP are restricted when dropout is applied.

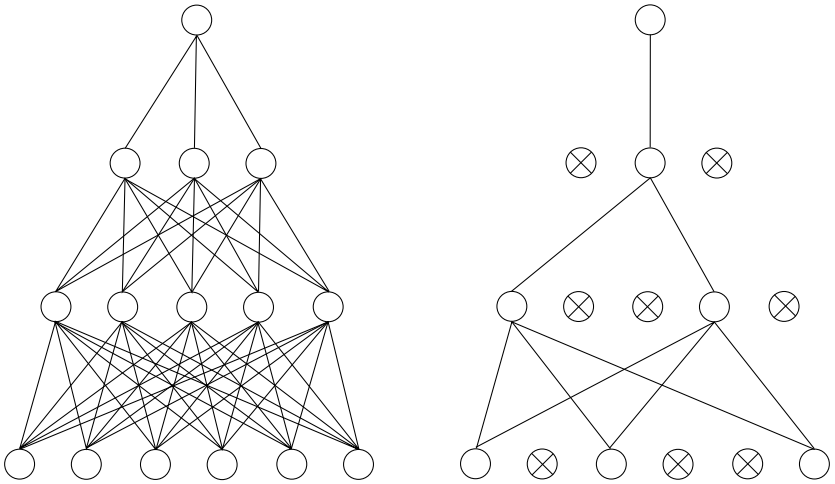


Figure 2.3: *A normal two layer fully connected MLP(left) and the same network with dropout applied (left). Crossed units have been dropped due to dropout.*

2.1.5 Convolutional neural networks

In images, each pixel provides an input to an artificial neuron in a fully-connected layer. Even for small images, a large number of weights are required to form a layer. With many weights, the networks are more liable to overfit on the data in the training set. Many weights also lead to increased memory consumption, which can be a limiting factor when the network is implemented on embedded hardware. LeCun *et al.* [30] showed that *convolutional neural networks* (CNNs) can be trained with backpropagation, which improve their ability to process high-dimensional spatial data in the form of images. CNNs are present in many current state-of-the-art approaches to computer vision problems such as semantic segmentation [32, 59] or image classification [28, 14].

CNNs offer an alternative architecture to the normal fully-connected neural networks that is biologically inspired by the neurons present in the visual pathways of animals [19]. Compared to the fully-connected neural network, a CNN is spatially invariant. It has a reduced number of parameters which requires less memory and makes them faster to train as well as being more robust to small deviations in the input. These properties are achieved by four architectural ideas: local receptive fields, shared weights, pooling and stacking of layers.

Artificial neurons in a convolutional layer perform the same kind of computations as in the fully-connected setting but they only consider inputs in small patches known as the local receptive field. The reasoning behind the local receptive field is that local groups of pixels in images tend to be highly correlated, forming distinct motifs that can be detected. The weights applied to a small patch are known as filters because they will only activate when specific features of the input are detected. Certain low-level features such as edges or colours can appear in multiple areas of an image. To make use of the spatial invariance of the features, neurons across the image share the weights to look for the same feature. By sharing the weights, the output of the weight-connected neurons are equivalent to performing a discrete convolution across the input.

Multiple neurons looking at the same local patch with different filters are stacked depth-wise and each filter treats the whole depth of the input within the local receptive field. Figure 2.4 shows how a square image in the form of an input-volume of size $l \times l \times d$ is transformed to an output-volume of size $l_f \times l_f \times N_f$ by applying N_f numbers of $F \times F \times d$ shaped filters. The spatial output size l_f of a convolutional layer is given by

$$l_f = (l - F + 2P)/S + 1 \tag{2.9}$$

and is a function of the spatial input size l, the filter size F, the amount of padding (width P of zeroes bordering the input) and the stride S (step size between considered inputs).

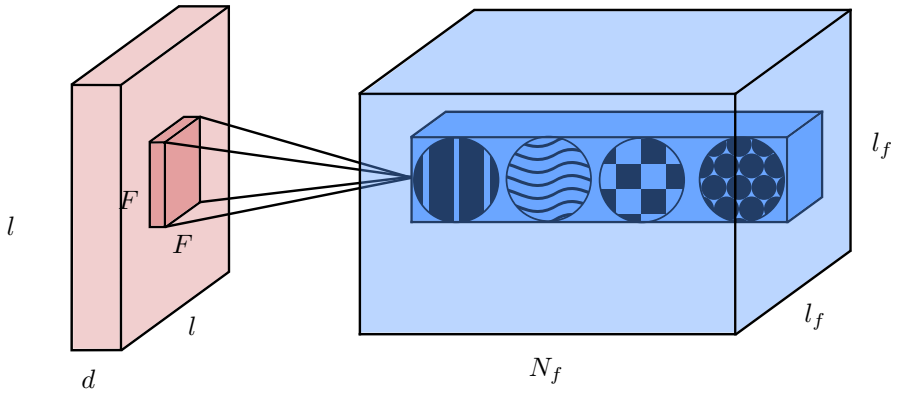


Figure 2.4: *A square input image of dimension $l \times l$ with d colour-channels (d=3 for RGB and 1 for grayscale) is passed through a convolutional layer with N_f number of $F \times F \times d$ shaped filters to produce an output of size $l_f \times l_f \times N_f$.*

2.2 Predictive gating pyramid

Models used for spatiotemporal predictions within the field of deep learning often rely on a hidden state that is updated temporally and used to predict the next input. A drawback of such models is the lack of transparency, meaning it is difficult to monitor what the model learns. Michalski *et al.* [40] introduced a bi-linear model which learns explicit transformations by utilising spatiotemporal synchrony. This section aims to explain the concept of spatiotemporal synchrony and as well as the different parts and their usage in the presented model. The complete predictive gating pyramid model is presented along with the proposed extension to enhance the spatial awareness of the model.

2.2.1 Motion estimation by synchrony detection

Encoding of motion has been shown to consist of two independent contributions, namely encoding of temporal synchrony and content invariance [25]. This factorisation can be achieved by using multiple filter pairs that each encode one possible transformation of a specific input. By combining multiple of these filter pairs, it is possible to encode additional transformations that also show some level of content invariance.

Assume that two filters $\vec{w}_1$ and $\vec{w}_2$ encode an orthogonal transformation in pixel space between two images $\vec{x}_1$ and $\vec{x}_2$. Orthogonal transformations encompass all possible permutations of an input, including spatial transformations such as translation and rotation. Translations in natural images are seldom completely orthogonal since it would for example require data disappearing to the right to re-emerge from the left and vice-versa.

It is possible to detect an orthogonal transformation L by using filters that are related to each other by the transformation

$$\vec{w}_2 = L\vec{w}_1. \tag{2.10}$$

If then

$$\vec{w}_2^T\vec{x}_2 = \vec{w}_1^T\vec{x}_1 \tag{2.11}$$

is true, $\vec{x}_2$ is the transformation L applied to $\vec{x}_1$ and is referred to as the "synchrony condition" [25]. A transformation is implicitly encoded by detecting synchrony between the two filter responses. To see why the condition implies the transformation, note that $\vec{x}_2 = L\vec{x}_1 \Leftrightarrow \vec{w}_2^T\vec{x}_2 = \vec{w}_2^TL\vec{x}_1$ which yields,

$$\vec{w}_2^T\vec{x}_2 = (\vec{w}_2^TL\vec{x}_1 = (\underbrace{L^T\vec{w}_2}_{\vec{w}_1})^T\vec{x}_1) = w_1^T\vec{x}_1. \tag{2.12}$$

The orthogonality of L ($L^T = L^{-1}$) was used in Eq. (2.10) to derive Eq. (2.12). A transformation L is therefore present when the filter responses of two filters that are related by L as in Eq. (2.10) are high. The response of a filter is content-dependent so that a filter pair encodes the transformation applied to a certain input. Content invariance can be achieved by adding an additional layer that combines different filter responses.

To detect synchrony, it is not possible to use synaptic summation with thresholding. The sum of $\vec{w}_1^T\vec{x}_1+\vec{w}_2^T\vec{x}_2$ would obtain a maximum for input that would match both the filters. However, it is seldom the case that two inputs match the filters completely but rather are they a superposition of multiple filter responses. Then, the threshold has to be low enough to capture all the non-optimal responses. Since the filter responses are content-dependent, it would not be possible to discern between a pair of filters where both filters have a 50% overlap with the input (motion is present) and a pair with one filter 100% overlapping and another not overlapping at all (motion is not present). Less overlap of the filters would require a lower threshold, which allows for more ambiguities with false positives.

A different approach to detect synchrony is to use multiplicative "gating" interactions between the filter responses. The product $p = \vec{w}_1^T\vec{x}_1 \cdot \vec{w}_2^T\vec{x}_2$ will attain a high value for two filters that both have a large positive (or negative) response. A small response of either $\vec{w}_1^T\vec{x}_1$ or $\vec{w}_2^T\vec{x}_2$ will effectively shut off the response of p so only filter-pairs encoding some part of the motion will contribute to the product. This dependence on both responses makes the gating similar to a logical "AND" operation, whereas the synaptic summation with thresholding is more akin to an "OR" operation [36].

Figure 2.5 shows three 1-dimensional cases of the product response of filters. When both inputs are well-represented by the filters, the product response will be high due to a high level of synchrony (Fig. 2.5b). When only one input matches a filter well, the product will be low due to the multiplication as opposed to summation (Fig. 2.5c). The input dependence of a filter pair is visible in Fig. 2.5d where an input is out of phase in relation to a filter, resulting in a product close to zero. By pooling over multiple filters related by the same transformation but different phases, content invariance can be achieved and results for not completely orthogonal transformations.

2.2.2 Relational autoencoder

The interest for gating mechanisms in artificial neural networks has increased in the last few years and found applications in a wide variety of fields, see [47] for an overview. Memisevic and Hinton [38] introduced gated bilinear models in the form of gated restricted Boltzmann machines to learn relations between images. The gated restricted Boltzmann machine is a probabilistic model that is trained by approximating the gradient through sampling. This work was extended to gated autoencoders in [37] where a latent variable z can learn to transform two inputs x and y into the other by,

$$z_k = \sum_{ij} W_{ijk}x_iy_j. \tag{2.13}$$

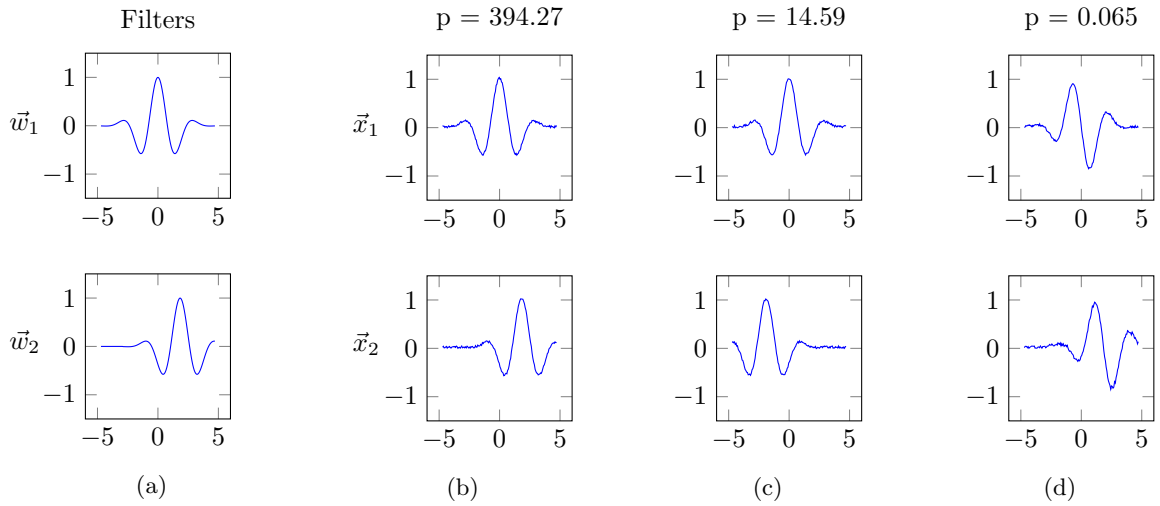


Figure 2.5: *Figure recreated from [25] showing different cases for the gating product* $p = \vec{w}_1^T\vec{x}_1 \cdot \vec{w}_2^T\vec{x}_2$. *Fig. 2.5a shows two filters* $\vec{w}_1$, $\vec{w}_2$ *related by a translation* L *to the right. If the inputs* $\vec{x}_1$ *and* $\vec{x}_2$ *are well aligned with the filters, there will be a high activation of* p *(Fig. 2.5b). Fig. 2.5c shows the case when* $\vec{x}_1 = \vec{w}_1$ *but* $\vec{x}_2 \neq L\vec{x}_1$ *and the response of* p *is greatly reduced. The final case shown (Fig. 2.5d) occurs when* $\vec{x}_2 = L\vec{x}_1$ *but* $\vec{x}_1$ *and* $\vec{w}_1$ *are out of phase by* $\pi/2$, *i.e. maximally unaligned.*

The autoencoder is trained by normal backpropagation, as explained in Section 2.1.2. Models that are linear in multiple inputs are known as bilinear models, as can be seen in the above equation. Each variable in the model modulate the connections between the others and can be thought of as gating the interaction,

$$z_k = \sum_j W_{jk}(x)y_j = \sum_i W_{ik}(y)x_i.$$

The model is symmetric around the interaction tensor W_{ijk} in the sense that, given any two of the variables it is possible to generate the third,

$$x_i = \sum_{jk} W_{ijk}y_j z_k \qquad y_j = \sum_{ik} W_{ijk}x_i z_k.$$

The model can learn to encode relations in z between two inputs x, y in an unsupervised fashion by training on the symmetric reconstruction loss,

$$\mathcal{L} = ||x - \hat{x}||^2 + ||y - \hat{y}||^2. \tag{2.14}$$

A drawback of the relational autoencoder is that the interaction tensor W_{ijk} contains $n_x \times n_y \times n_z \sim \mathcal{O}(n^3)$ parameters leading to large memory consumptions for high dimensional inputs such as pictures.

2.2.3 Factored gated autencoder (GAE)

By factoring the interaction tensor into the "inner product" of three matrices [39], the encoding of motion can be disentangled into detection of synchrony and encoding of content invariance as presented in Section 2.2.1. The factorisation produces three matrices of size $F^X : n_x \times n_F$, $F^Y : n_y \times n_F$, $F^Z : n_z \times n_F$ with $n_x = n_y$ being the input size, n_F being the number of filters and n_z being the number of hidden units,

$$W_{ijk} = \sum_{f=1}^{n_F} F^X_{if}F^Y_{jf}F^Z_{kf}.$$

This factorization leads to a reduced memory consumption scaling as $\mathcal{O}(n^2)$. A latent variable z in a *factored gated autoencoder* (GAE, [37]) is then given by,

$$z_k = \phi(\sum_f F^Z_{kf}\left(\sum_i F^X_{if}x_i\right)\left(\sum_j F^Y_{jf}y_j\right)). \tag{2.15}$$

F^X and F^Y are matrices with corresponding columns being filter pairs for detecting synchrony as mentioned in Section 2.2.1. F^Z is a matrix that learns to combine the filter pairs to encode content invariant transformations. Since the hidden units learn to encode transformations independent of input, they capture the structural dependencies of the transformation. Michalski *et. al.* [40] compare the structural encoding to the structural role of grammar in languages and refer to the hidden units as mapping units or grammar cells. The result of rewriting Eq. (2.15) as matrix-vector products yields

$$z = \phi(F^Z \left(F^X x \cdot F^Y y\right)). \tag{2.16}$$

To reduce the number of superscripts and to use the same notation as presented in [40], the parameters are renamed as: $z \to m$, $F^Z \to W$, $F^X \to U$, $F^Y \to V$ and for video sequences, the inputs are numbered as $x^{(i)}$. Equation (2.16) is then rewritten as

$$m^{(1:2)} = \phi(W \left(Ux^{(1)} \cdot Vx^{(2)}\right)). \tag{2.17}$$

Given mapping unit activations m and one of the observations, it is possible to reconstruct the other using

$$\begin{cases} \hat{x}^{(2)} & = V^T \left(Ux^{(1)} \cdot W^T m^{(1:2)}\right) \\ \hat{x}^{(1)} & = U^T \left(Vx^{(2)} \cdot W^T m^{(1:2)}\right). \end{cases} \tag{2.18}$$

This is equivalent to applying the transformation encoded in m to one of the inputs to get the other. An efficient encoding of transformations can then be learned by minimising a symmetric reconstruction loss. The symmetric loss is defined by the *mean-square error* (MSE) with C being a normalising scalar value,

$$\mathcal{L} = \frac{1}{C}\left(||x^{(1)} - \hat{x}^{(1)}||^2 + ||x^{(2)} - \hat{x}^{(2)}||^2\right). \tag{2.19}$$

For the MSE, C will be equal to the number of inputs, *i.e.* $2 \cdot dim(x)$ in the equation above. It is possible to use other norms and loss functions instead of the MSE in Eq.(2.19).

2.2.4 Predictive gating pyramid (PGP)

Encoding a mapping between two inputs can be seen as performing a first-order Taylor approximation of the input [40]. With this view, the mapping units model the partial first-order derivative of the input with respect to time. The mapping units of the GAE learn to encode structure of transformations, not content. Memisevic and Hinton [39] showed that encoding a transformation between two images and then applying it to another image would infer the same transformation, despite having different content. This content invariance can be used to predict future inputs of a time-series. If the transformation $m^{(1:2)}$ encoded between $x^{(1)}$ and $x^{(2)}$ according to Eq. (2.17), is assumed to be approximately constant in time, it is possible to predict $x^{(3)}$ by applying the encoded transformation $m^{(1:2)}$ to $x^{(2)}$,

$$\hat{x}^{(3)} = V^T \left(Ux^{(2)} \cdot W^T m^{(1:2)}\right). \tag{2.20}$$

A visualisation of the prediction can be seen in Fig. 2.6a. Training to minimise the predicted loss defined as

$$\mathcal{L} = \frac{1}{C}\left(||x^{(3)} - \hat{x}^{(3)}||^2\right), \tag{2.21}$$

improves the predictive abilities of the model and is done by unrolling the computational graph in time to facilitate backpropagation through time [57]. The loss in Eq. (2.21) is defined using the MSE with $C = dim(x)$ but other loss functions are also possible.

As long as the transformation is assumed constant, further predictions can be inferred iteratively to make multi-step predictions,

$$m^{(2:3)} = \phi(W \left(Ux^{(2)} \cdot V\hat{x}^{(3)}\right)), \qquad \hat{x}^{(4)} = V^T \left(U\hat{x}^{(3)} \cdot W^T m^{(2:3)}\right). \tag{2.22}$$

The loss to be minimised for multi-step predictions is

$$\mathcal{L} = \frac{1}{N \cdot C} \sum_{i=1}^{N} ||x^{(t+i)} - \hat{x}^{(t+i)}||^2. \tag{2.23}$$

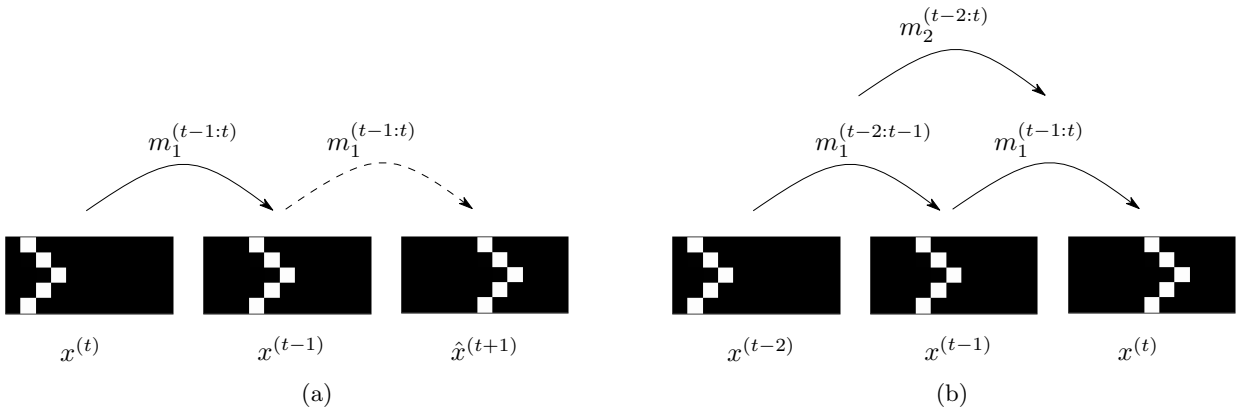


Figure 2.6: *Visualization of a one-layer PGP used for prediction by assuming the first layer mapping is constant (2.6a) and a two-layer PGP learning a mapping between mappings (2.6b).*

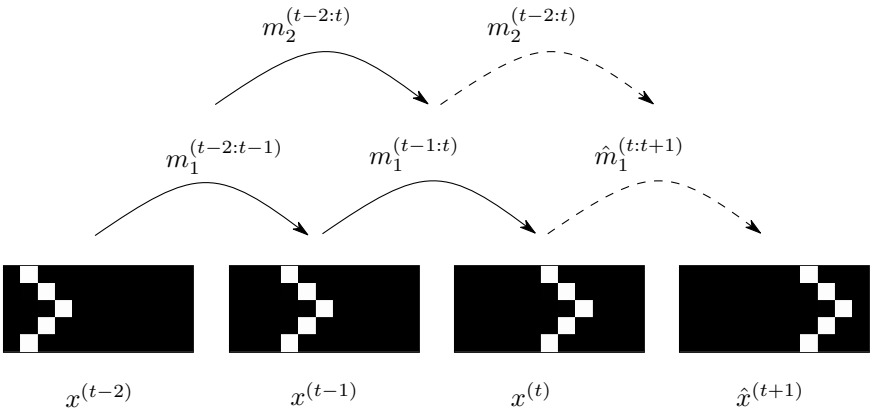


Figure 2.7: *A two-layer PGP used for predicting the next input in a time-series. The two-layer model assumes m_2 is constant between the frames and infers an approximation of $m_1^{(t:t+1)}$ to be used for predicting the next input $x^{(t+1)}$.*

It is an analogous extension of the one-step predictive training criterion seen in Eq. (2.21). using the MSE with N being the number of predicted time steps and C a normalising constant, often chosen to be the dimension of the input x. The training criterion in Eq. (2.23) allows bilinear models to be used as generative models for sequences. By training on the symmetric reconstruction in Eq. (2.19) and prediction in Eq. (2.23), the GAE will learn a representation of the manifold along which the input evolves in time.

If the transformation between input data is not constant in time but contains some form of acceleration, it is possible to model higher-order transformations by stacking GAEs to model transformations between transformations. This is analogous to approximating the second derivative by taking the derivative of the first-order derivative [40, 39]. The mapping unit activations are fed into a second GAE to model a higher-order transformation,

$$m_2^{(t-2:t)} = \phi(W_2 \left(U_2 m_1^{(t-2:t-1)} \cdot V_2 m_1^{(t-1:t)} \right)). \tag{2.24}$$

The encoded higher-order mapping can then be used to infer a new low-level mapping (Eq. (2.25)) which in turn is used to infer the next input (Eq. (2.26)).

$$\hat{m}_1^{(t+1)} = V_2^T \left(U_2 m_1^{(t-1:t)} \cdot W_2^T m_2^{(t-2:t)} \right) \tag{2.25}$$

$$\hat{x}^{(t+1)} = V_1^T \left(U_1 x^{(t)} \cdot W_1^T \hat{m}_1^{(t:t+1)} \right) \tag{2.26}$$

Stacking GAEs in this pyramid-like structure for prediction creates a *predictive gating pyramid* (PGP) [40]. The depth of the PGP determines which entity relating to the derivative with respect to time is assumed constant; velocity for one layer, acceleration for two layers *etc.* Figure 2.7 visualises the calculations performed for prediction with a two-layer PGP.

2.2.5 Convolutional PGP

The idea of this thesis is to extend the PGP to operate with convolutions in order reduce the number of parameters and learn spatial transformations that can be applied to more high-dimensional data. By using convolutions, the network learns local transformations that are spatially invariant and does not have to rely on global filters. In the convolutional version of the PGP, all of the matrix-vector products in Eq. (2.17) are replaced with convolutions resulting in

$$m^{(t-1:t)} = \phi(W * \Big(U * x^{(t-1)} \cdot V * x^{(t)}\Big)), \tag{2.27}$$

with $*$ representing the discrete convolution operation.

The terms of the above equation are analogous to the fully-connected case, except for U, V and W now being stacks of filters convolved over their respective input, as explained in Section 2.1.5. Note that U and V need to have the same number of filters and output dimensions but can have filters of different sizes. The convolutional analog of the transposed matrix multiplications for the reconstruction and prediction in Eq. (2.18) and Eq. (2.20) is the transposed convolution [7]. In the case of unit stride the transposed convolution is equivalent to the cross-correlation which is a convolution with flipped filters. For non-unit stride, the convolution is dilated by additional zeros between the values so as to recover the proper input size of the corresponding convolution.

3 Data and training

The chapter will provide information on the datasets used and how training of the models was performed. The models were built by means of the Python package Lasagne which uses Theano [1] as a back-end for the computations and training of parameter setting. Theano uses automatic differentiation of mathematical expressions which simplifies training of neural networks with gradient-descent.

3.1 Dataset

Two datasets were used to analyse and evaluate the two models. A one-dimensional dataset consisting of sequences of binary numbers was used to analyse the performance of the convolutional version and to find appropriate parameter values. A single "1" is moving around on a background of "0"s. The velocity, *i.e.* the distance covered between subsequent frames, is uniformly sampled in the range $v \in [-v_{max}, v_{max}]$, $v \in \mathbb{Z}$. Each sequence consisted of 3 frames with the "1" being displaced by v relative to the preceding frame with a starting point chosen so the "1" would never leave the frame. Due to the spatial locality of the convolution, the overall size of each frame just had to be larger than $2v_{\max}$. Usually a size of 32 and $v_{\max} = 6$ was used. Figure 3.1 shows an example sequence of three frames with a binary "1" moving at a speed of $v = 3$.

Figure 3.1: *Example sequence for the one-dimensional dataset.*

A more complex and intuitive dataset for modelling motion is the 2-dimensional dataset of bouncing balls [50]. Compared to the 1-dimensional dataset, the objects now have a shape and the movements are not restricted to left or right, as seen in Fig. 3.2. The original version of the dataset contained binary video sequences of 2 balls interacting in a box of size 30×30 pixels. A larger box of size 64×64 pixels was used in the experiments. The default radius for the balls during training was $r = 8$ pixels. The models were tested on balls of different sizes and varying numbers of balls. Each generated sequence consisted of 20 frames of size $1 \times 64 \times 64$, corresponding to black and white frames of size 64×64 pixels. The training set consisted of 15 epochs of 32000 sequences each. A validation set and a test set each consisting of 32000 sequences was used for early stopping and evaluation of the models.

3.2 Training

For the fully-connected model, it was necessary to pretrain the model first on reconstructing the inputs (Eq. (2.19)) in order to initialise the weights for the predictive training (Eq. (2.20)), [3, 40]. The same procedure showed improvements when applied to the convolutional version. The models are trained to produce the pixel-intensities that are as close as possible to those of the time series found in the dataset. For the conducted experiments, no preprocessing of the data was performed [40]. Some experiments of normalising the input data to mean 0 and unit variance were performed but no noticeable improvement for either convolutional or fully-connected version was detected.

Each training step consisted of loading a batch containing a 5D tensor of size: (`batchSize` × `sequenceLength` × `imageChannels` × `imageWidth` × `imageHeight`). Each batch would then be split and copied into a new tensor

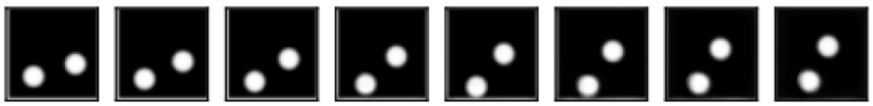


Figure 3.2: *Sequence of 8 frames of size* 64×64 *from the two-dimensional dataset of bouncing balls.*

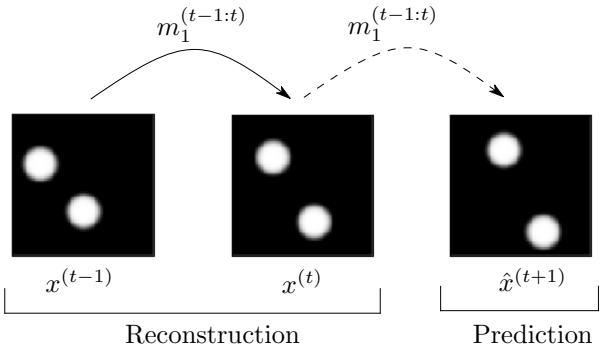


Figure 3.3: *In the case of a one-layer PGP, two distorted inputs from a sequence* $x^{(t-1)}$ *and* $x^{(t)}$ *are used to encode a transformation* $m_1^{(t-1:t)}$ *with Eq.* (2.17)*. During the pretraining, the mapping is used to reconstruct the two undistorted inputs according to Eq.* (2.19)*. The loss is the MSE between the two undistorted inputs and the two reconstructions. The predictive training uses the mapping to predict the next entry* $\hat{x}^{(t+1)}$ *in the sequence according to Eq.* (2.20)

so the entire sequence could be used during training. For reconstruction, the new tensor would have shape ($\texttt{batchSize} \cdot \texttt{sequenceLength}/2 \times 2 \times \texttt{imageChannels} \times \texttt{imageWidth} \times \texttt{imageHeight}$). The first dimension in the training data was shuffled and split into smaller batches of size $\texttt{miniBatchSize} = 32$ which were provided as input to the network. The network was then trained to minimise the MSE of the reconstruction or prediction.

The fully-connected model was pretrained to reconstruct two input frames that have been corrupted by Gaussian noise, as seen in Fig. 3.3. A dropout factor of 0.3 (Section 2.1.4) was applied to the filter responses to encourage the usage of multiple filters and to make the model more robust to deviations from the training data. The convolutional model was trained with L_2-regularization and additional noise for the pretraining. Since the convolutional model learns local transformations, it required a modification of the two-dimensional dataset to go through the same training as the fully-connected version. A border of 2 pixels width with value 1 was added around the training data so a wall would be present to bounce against. The spatial dimension of a frame was increased from 64×64 pixels to 68×68 pixels with the appended surrounding pixels having value 1.

Lasagne offers multiple algorithms to train neural networks with gradient-descent. In the original work of Michalski *et al.* [40], the PGP was trained with stochastic gradient-descent with momentum (Section 2.1.2). Due to fast convergence and less sensitivity to hyperparameter choice, the Adam algorithm [24] was used instead for training. A learning rate of $\eta = 10^{-3}$ was used for reconstruction and 1-step predictive training. The learning rate was lowered to $\eta = 10^{-4}$ for 2-step predictive training. All 2-step predictions were preceded by a 1-step prediction so that training the model afterwards focuses on minimising the 2-step prediction.

4 Results and discussion

The convolutional PGP has been applied to a one-dimensional dataset to test the functionality of the developed model. Then, the convolutional and the original fully-connected versions were evaluated on a two-dimensional dataset. The performance of both models are compared and discussed. The chapter is concluded by presenting some problems of the dataset and training, and suggests possible venues for future research.

4.1 One-dimensional data

To get some intuition for the convolutional PGP and appropriate parameter settings, the model was evaluated on a dataset containing binary videos of moving "1"s as presented in Section 3.1. The goal was to study the importance of different filter sizes in the model. The filter sizes of U (F_U) and V (F_V) limits the maximum velocity v the model is able to reconstruct, since synchrony between the filters needs to occur. Experiments were carried out to determine how the filter size of U, V and W affect the ability to predict sequences.

A one-layer convolutional PGP can reconstruct inputs with velocities that satisfy

$$|v| \leq \frac{F_U - 1}{2} + \frac{F_V - 1}{2} \tag{4.1}$$

regardless of the size of W. Using untied weights, *e.g.* introducing new filters for the transposed operations in Eq. (2.19) yields the same results. The right-hand side of Eq. (4.1) is the maximum distance for which synchrony can occur for a given F_U and F_V with F_U, $F_V = 2i + 1$, $i = 1, 2, \ldots$. The filters emerging for the reconstruction objective of this dataset with applied L_1-regularization are intuitive and interpretable, as can be seen in Fig. 4.1. The model was trained on velocities $v \in [-3, 3]$ which contain seven different integer velocities, all of which are found encoded in the filters. The used filter sizes are listed in Tab. 4.1. Each non-empty filter

Layer1	U	V	W
Size	5	5	1
#	8	8	10

Table 4.1: List of parameters used for the one-dimensional example with a 1-layer PGP.

pair in Fig. 4.1 encodes one of these velocities with the encoded motion written above. In this setting, it was possible for each mapping unit to explicitly encode one type of motion but usually one transformation required multiple mapping units that each combined multiple filter responses. It is possible to use fewer filters than possible velocities but each filter pair then has to encode multiple movements and interpreting the encoding of transformations is not as transparent.

Experiments to determine the size dependence of U and V for the prediction task were performed by training models with varying filter sizes of U, V and W. Figure 4.2 shows the predicted MSE of a one-layer convolutional PGP for different sizes as a function of velocity. From the results in the left graph of Fig. 4.2, we see that a symmetric size of the filters in U and V had the best average performance. The right figure shows that using a W of size larger than 1 improved the ability to predict velocities. However, it also reduced the effect and meaning of the synchrony condition because of multiple contributions to the mapping unit.

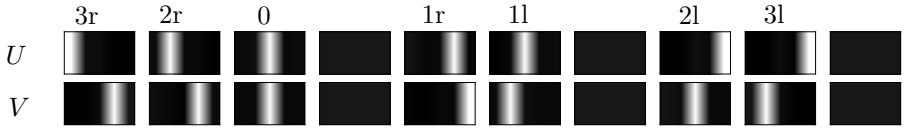


Figure 4.1: *Filters for the 1D data with $v \in [-3, 3]$ pixels after training on reconstruction. There are seven possible velocities and the filters accurately capture each motion. The filters and number of mapping units are listed in Tab. 4.1. The sparse filters are a side effect of using L_1-regularization during pretraining.*

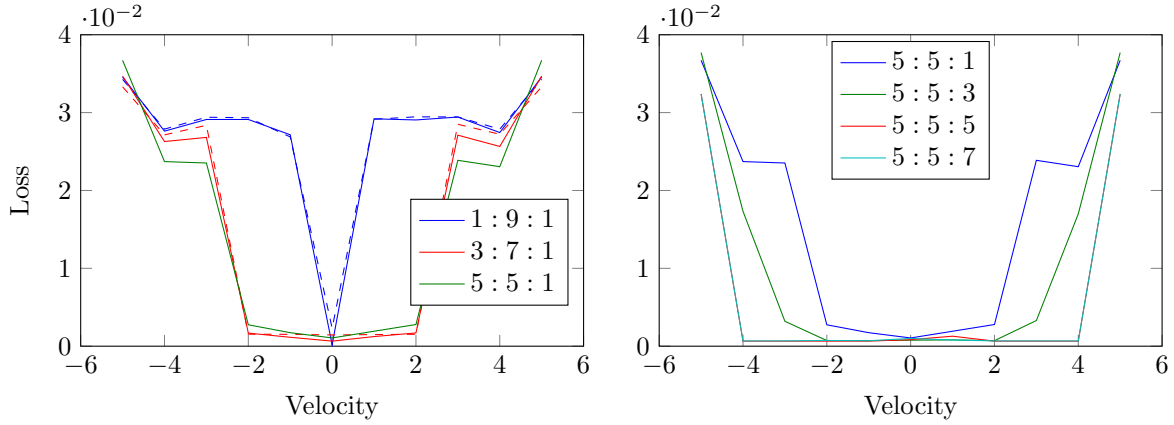


Figure 4.2: *Predictive abilities measured by MSE of the model with fixed receptive field of W but varying size of U and V (left) and varying size of W but fixed for U and V (right). The left image shows the loss for different velocities for architectures of size $U : V : W$. Dashed lines represent the same setup as the same-coloured counterpart but with U and V size flipped, i.e. $V : U : W$. The measured loss is the MSE of the one-step prediction defined in Eq.* (2.21)*. The model behaved similarly for mirrored filter sizes and performed the best for symmetric filter sizes. The right image shows the predictive abilities of the model with a symmetric size of* 5 *for U and V with varying size of W. A W larger than the receptive field of U and V did not improve the prediction. This is likely due to the fact that the transformations are supposed to occur within the visibility of U and V for synchrony to be possible. By using a larger size of W, other transformations are also taken into account which could lead to ambiguities in a less sparse environment.*

4.2 Two-dimensional data

In order to evaluate the predictive capabilities of the convolutional PGP, it was compared to the original fully-connected version on the task of predicting sequences of bouncing balls as described in section 3.1. For both models, it was important to use untied weights to achieve good predictions and to also generalise beyond the trained number of predictions. For the majority of the experiments, no preprocessing of the input data was performed since good results can be obtained without it [40, 50]. Some experiments were done by subtracting the mean from the input data and dividing by the standard deviation. However, neither of the models showed any significant improvement by this preprocessing.

4.2.1 Original PGP

The fully-connected model uses the whole frame as input and needs to learn transformations at every point in the input space resulting in many learnable parameters and a long time to train. These properties made it difficult to train the model to achieve good results. Figure 4.3 shows an example sequence of a fully-connected version trained on 2-step prediction. The model generating the sequences was a 2-layer PGP with (512, 256) filters and (256, 128) mapping units per layer with a sigmoid nonlinearity. To facilitate the slow evolution of top-layer mappings, the average of two top-layer mappings were used to generate the next frame in the sequence [40].

Pretraining the fully-connected model with regularization was necessary for good performance. The pretraining was performed by corrupting the input with Gaussian noise and minimising the de-noised reconstruction loss in Eq. (2.19). Applying dropout to the combined filter-response term $(Ux^{(t-1)} \cdot Vx^{(t)})$ in Eq. (2.17) forced the model to use multiple filters to reconstruct the input, making it more robust to deviations in the input. Combining dropout with the de-noising criterion during pretraining provides a good initialisation for the predictive training. The filters emerging after predictive training then contain a location and a translation component, as can be seen in Fig. 4.4.

The objective of the pretraining was to provide the filters with spatial locality by making a large part of each filter have the value 0. Apart from the de-noising criterion, this can be achieved by additional L_1- or L_2-regularization. Adding additional weight regularization generally lead to few active and global filters (see Fig. 4.5) depending on the regularization strength. The predictions by models with regularized weights during pretraining were generally coarser and inferior to de-noising models.

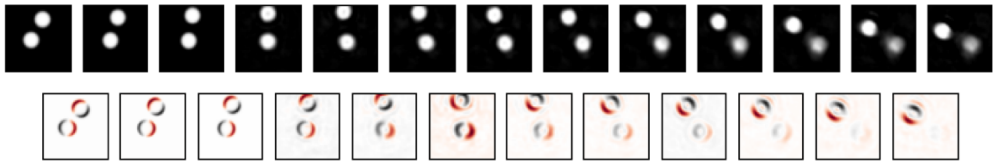


Figure 4.3: *Sequence of bouncing balls generated with a fully-connected PGP with (512, 256) filters and (256, 128) mapping units respectively per layer and average top-layer mappings. The first four frames are seeded to the model. To easier visualise the movements of the balls, the second series contains the difference between each pair of successive frames above. The predictions are a bit blurry, possibly due to the averaging procedure introduced by the MSE [34].*

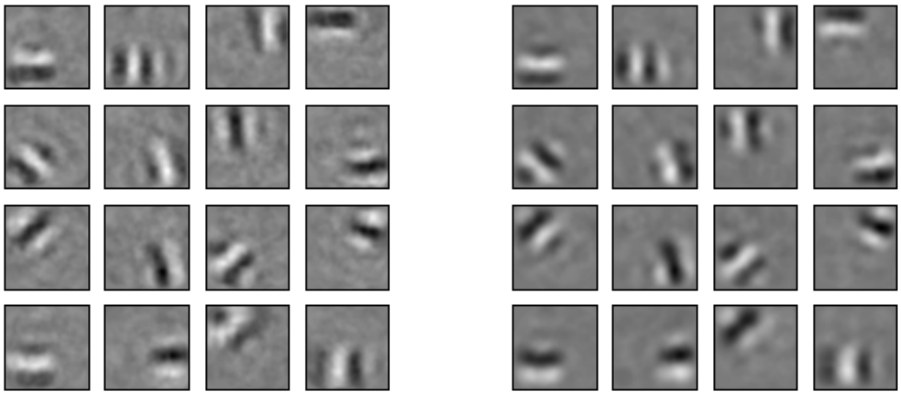


Figure 4.4: *Example of filters U (left) and V (right) for the fully-connected version of a two-layer PGP trained on bouncing balls and pretrained with de-noising reconstruction as regularization. White parts correspond to positive weights in the filter and black to negative. Because the model has to learn movements in every direction at every point, it requires many filters. The data has a high resolution of* 64×64 *pixels, so each filter in the first layer contains* 4096 *weights leading to long training times.*

It is possible that models with weight-regularized pretraining would reach the same performance as de-noising models with optimal values of the regularization strength. De-noising the input provides a more lenient form of regularization since the parameters are not continuously guided towards 0 but try to remove the presence of noise by reducing the parameter values towards 0. The de-noising criterion is therefore not as sensitive to optimal values of the regularization hyperparameter, making training easier and reproducible.

Michalski *et al.* [40] suggested using the sigmoid nonlinearity in Eq. (2.17) which squashes the values to $(0, 1)$. The values of the mapping units can then be interpreted as a probability of the encoded motion being present. Using the sigmoid as nonlinearity seems a bit counter-intuitive since the function is supposed to never reach 0, so each mapping unit is always active. In practice, the activation can reach 0 for very large negative values due to rounding or when explicitly coded. When no transformations are encoded by a mapping unit, the value of the activation would be 0.5 when passed through the sigmoid. This leads to relatively large negative biases which inhibit the learning of autoencoders [27].

Experiments were carried out to test if sparse activations would improve the results. The sigmoid nonlinearity was then exchanged with other nonlinearities such as the ReLU and the truncated rectifier [27]. The additional sparseness invoked by both versions of the rectified nonlinearity often lead to few mapping units connected to many filter pairs, and therefore encoded many transformations. The predictions and reconstructions were coarser and worse compared to models with the sigmoid. A possible explanation for this behaviour could be that, by using untied weights, the GAE consists of an explicit encoding and decoding part. By using the sigmoid as activation function, each mapping unit is partially active, leading to more information for the decoding part to use. The decoding part then gets more information than is actually necessary and can learn which activations are relevant. More of the filters can then be used to get better predictions.

To test the model, the number of filters and mapping units for both layers were varied in powers of 2. For the first layer, the number of filters in U and V were varied in the ranges of $[256, 1024]_1$ and the number of mapping units in $[128, 512]_1$, with the subscript indexing the layers. The number of filters and mapping

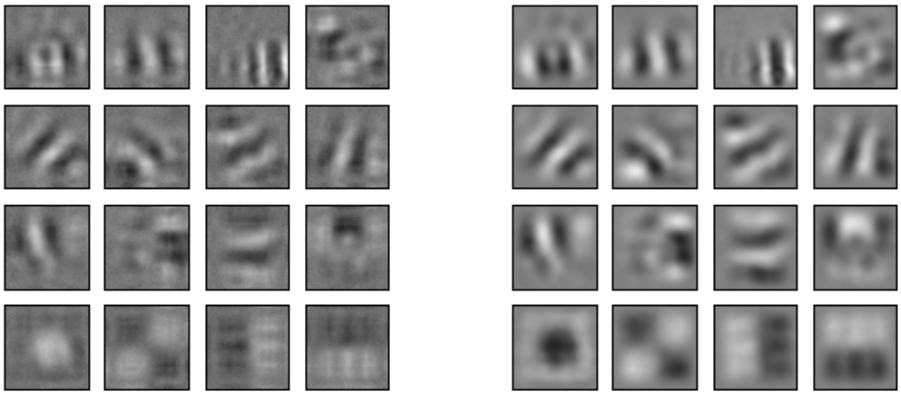


Figure 4.5: *Example of filters U (left) and V (right) for fully-connected version of a two-layer PGP pretrained without regularization. Most filters encode close to global motion and some (last row) seem to provide information for location of transformation. Similar filters emerge when pretraining with weight regularization and non-optimised regularization strength.*

units for the second layer were correspondingly varied in $[128, 512]_2$ for U and V filters and $[64, 256]_2$ for the mapping units. Using more parameters than 512 filters, 256 mapping units in layer 1 and half of the numbers in the second layer rarely improved the 2-step prediction. A large part of the first layer filters were empty with 512 filters in the bottom layer but it still outperformed a model with half of the number of parameters on minimising the 2-step MSE.

Some training steps that improved the performance of the fully-connected model was to first pretrain the model with MSE on reconstruction with added Gaussian noise. The model was then trained on prediction up to two frames ahead with either Huber loss or MSE, combined with a weak or zero-valued L_2-regularization term. Adding noise to the input during predictive training helped as well but it is likely that a smoothing of the input data would be better. The is that noise is applied everywhere in the frame but the model mainly has a tendency to blur the balls. Other training criteria with potential improvements is to penalise small non-zero gradients in the output, or train the model like a generative adversarial network with an additional discriminator network [11].

4.2.2 Convolutional PGP

The convolutional equivalent of the fully-connected PGP uses a filter size of 1×1 for the W, only considering local transformations within the receptive field of U and V. As with the fully-connected version, it was pretrained on reconstructing the input to initialise the weights for the predictive training. The model does not depend as much on the pretraining step as the fully-connected version, but generally it would improve the performance.

The first experiments were conducted with a one-layer convolutional PGP. The model consisted of 24 filters of size 13×13 and 12 mapping units of size 1×1. Intuitive filters encoding translations appeared when the model was trained on prediction, see Fig. 4.6. However, the one-layer model had problems maintaining the shape of the balls and the balls would quickly disappear in the predictions. Training on longer sequences only marginally improved the performance. As with the one-dimensional dataset, the predictive capabilities improved by using mapping units with a larger filter size but it also produced less interpretable filters.

To remain consistent with the fully-connected PGP, a two-layer model was introduced. The default version of the convolutional PGP, for which most results are reported, contained two layers with parameters listed in Tab. 4.2. Since the convolutional version only considers a local area for a transformation, it was necessary to include borders around the area where the balls should bounce, as explained in Section 3.1. Otherwise, there would be no information indicating a bounce against the image contour and the balls would just disappear out of frame. By including the borders in the predictive training, the model focused more on placing the borders correctly instead of moving the balls, leading to poor predictions. To overcome the error, a border of 2 pixels width was added around the frames but the loss was defined on the content within the border. The model then had to learn bouncing against the edges as well as move the balls. This way, the same training and validation set as for the fully-connected model could be used.

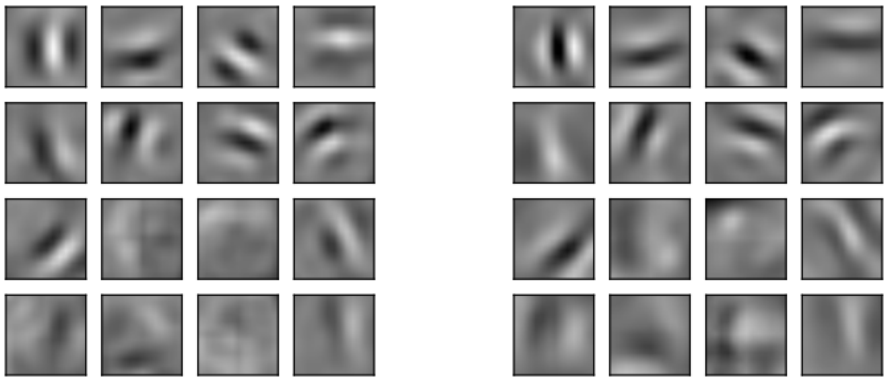


Figure 4.6: *Filters for the one-layer convolutional PGP with 24 U and V filters of size* 13×13 *and 12 W filters of size* 1×1*. The model was trained on 1-step prediction with L_2-regularization, tied input weights and no borders were present. The model learns intuitive filters for translation but cannot handle bouncing against the image contour since it does not detect anything to interact with.*

Layer1	U	V	W
Size	13	13	1
#	36	36	10
Layer 2	U	V	W
Size	7	7	1
#	16	16	5

Table 4.2: Table of parameters for an often used setup of the two-layered convolutional PGP applied to the bouncing balls.

Untied weights are often necessary for the convolutional model to perform well. Similarly to the original model, the filters then become closely tied because of the synchrony condition. Due to the fewer parameters found in the convolutional model, it was faster to train and not as dependent on reconstructive pretraining before the predictive training. Pretraining the model still improved the results but not to the same extent as in the original fully connected model. Opposed to the original model, it was beneficial to include weight-decay in the loss, and L_2-regularization yielded better filters over L_1. Exchanging the sigmoid nonlinearity to the ReLU further improved the performance of the convolutional model. The reason is likely due to the locality of the transformations in the convolutional model. With the ReLU, a mapping unit only activates where synchrony occurs and remains zero everywhere else.

Pretraining the convolutional version on reconstruction with MSE as the loss function and L_2-regularization followed by predictive training with L_2-regularization, yielded networks that could generalise beyond the training horizon. The sequence seen in Fig. 4.7 shows a predicted series of a network with parameters from Tab. 4.2. The network was trained on 2-step prediction and seeded with the first 3 frames from a previously unseen test set.

The training of a model with filters of size 1×1 for W and untied weights often reached a stable equilibrium that could generalise predictions further than the 2-steps it was trained on. A training procedure which often yielded good results was to pretrain the model with L_2-regularization on reconstruction with MSE as loss function and then train on prediction with either MSE or Huber loss, without regularization. Pretraining on MSE instead of Huber focuses the training on minimising the training sequences that produce a high loss. Many sequences then resulted in a low loss which provided a good initialisation for the predictive training. Performing predictive training with either MSE or Huber as loss function produced good models but it was not investigated in-depth which loss performed the best on the dataset.

To see how well the trained convolutional model generalised to similar data, the model was tested on datasets containing balls of different sizes (see Fig. 4.8) as well as a different number of balls (see Fig. 4.9). The smaller balls in Fig. 4.8 tend to be stretched and attenuated towards the size of the trained balls. Similarly, the larger balls shrink towards the optimal size learnt. The behaviour seen in Fig. 4.8 is similar to the behaviour of overfitting to the training data. The model was trained to minimise the predictive error of two balls of a specific size and the results are therefore to be expected.

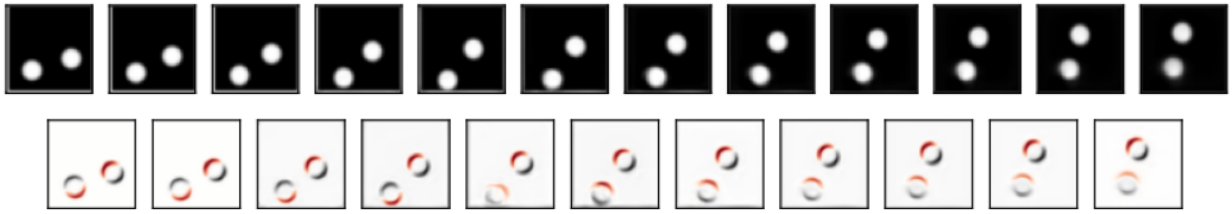
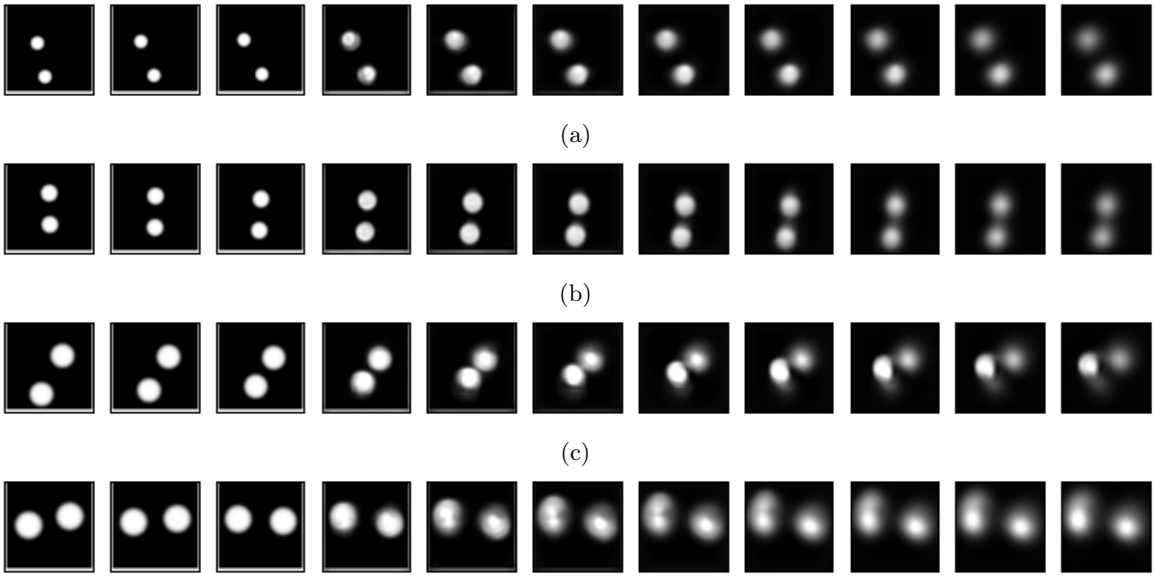


Figure 4.7: *The top row shows the predicted series of convPGP with parameters seen in Tab. 4.2 when seeded with three frames from a previously unseen test set. The model was trained on 2-step prediction but can accurately predict further ahead. The bottom row shows the difference between two successive frames with red indicating positive values and gray negative.*



(a) (b) (c) (d)

Figure 4.8: *ConvPGP with parameters from Tab. 4.2 trained on 2-step prediction applied to balls having an altered radius r of $0.67r$ (a), $0.83r$ (b), $1.17r$ (c), $1.33r$ (d). The network has overfitted the learned transformations to a specific size of the ball. Smaller radii mean less information for the model and the balls disappear quickly. For larger radii, the balls tend to shrink down to the learned size.*

The convolutional model showed good predictive performance when seeded with frames of three balls, see Fig. 4.9. The model was trained on 2-step prediction of two balls and could generalise transformations to three balls of the same size. Artificial neural networks and other machine learning models, tend to develop a strong bias for the training set by overfitting to it. The network can then have trouble handling data from outside the distribution of the training set, such as more objects [48, 22].

4.2.3 Comparison

To compare the performance between the convolutional and the fully-connected model, one version of each model that performed well was evaluated on a separate test set of bouncing balls. Both models were trained on 2-step prediction to minimise the MSE. A graph over the MSE for both models when predicting trajectories from an unseen test set, can be seen in Fig. 4.10. Neither of the models underwent any explicit parameter optimisation and the result is therefore only qualitative. The graph shows that the convolutional model performed better when predicting ahead.The fully-connected model consisted of (512, 256) filters and (256, 128) mapping units per layer which is approximately $6.8M$ parameters. The convolutional model had the setting listed in Tab. 4.2, adding up to approximately $43k$ parameters. The quotient of parameters between the two models was $6 \cdot 10^{-3}$. Both models hade more parameters than they required and they would likely perform better with an optimal learning rate and regularization strength.

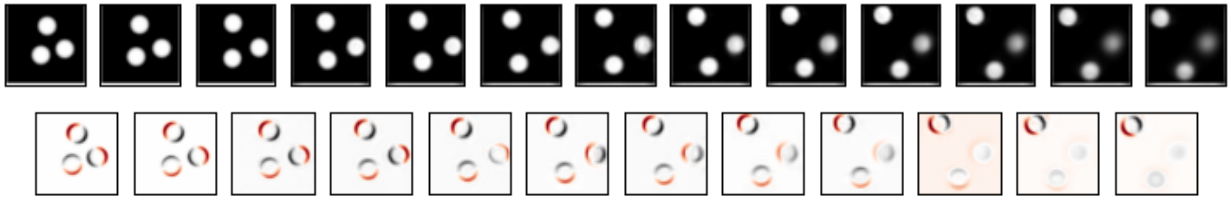


Figure 4.9: *The top row shows the predicted series of the convPGP when seeded with three frames containing three balls. The model was trained on 2-step prediction with two balls of the same size and still manages to simulate the movement of three balls a few steps ahead. The bottom row shows the difference between two successive frames with red indicating positive values and grey negative. The parameters used are listed in Tab. 4.2. The model was trained with Huber loss and L_2-regularization.*

Michalski *et al.* [40] suggested using the average top layer mapping when predicting ahead to assume a slowly changing top layer transformation. Average top layer activation was used in the experiments of this thesis for the fully-connected model with good results. The convolutional model did not benefit from the averaging. The reason behind it is likely that a mapping in the convolutional model is spatially local and is not present at the same location at two consecutive time steps. The original model uses the sigmoid nonlinearity which always produces an activation. For the convolutional model, which learns local transformations, it is better to use the ReLU so many of the mapping unit activations are 0 across the frame.

A big difference between the two models lies in what kind of dynamics they have learned. The fully-connected version does not require any border around the frame for a ball to bounce, because it has learned that a bounce always takes place at the image border. The model has learned a manifold for the pixels in the system and how they light up, which are specific for balls of a certain shape and velocity range in one particular bounding frame. The model would have to be retrained if the same dynamics is expected to occur in a differently sized frame. The convolutional model is more general in the sense that it has learned to move balls of a specific size and range of velocity. The learned filters can then be used if balls of the same size are moving around in a differently sized frame. If no border is used for training or testing, the balls will not bounce at the edge and just continue moving out of frame, see Fig. 4.11.

The convolutional model could accurately predict the movements of the balls as seen in Fig. 4.10. For some sequences in the test set that contained many interactions between objects, the convolutional model could only predict a few steps ahead before generating frames that were completely white. The resulting losses of these cases were orders of magnitude larger than the normal loss, making comparisons between the models infeasible. There are multiple factors that could contribute to this behaviour. One such factor is that the predicted output was linearly mapped to the input space, *i.e.* no squashing or thresholding of the values took place.

Since there were few filters being used in the convolutional model, the same filters were likely to be applied over and over when a sequence was predicted. If such a pair of filters magnified the values of the balls, repeated application of the filters would lead to values of the balls exponentially increasing. More of the same transformation encoded by the filters would then be present, and the model would iteratively apply the same filters again. The resulting loss would then rapidly increase.

One way to approach this problem is to pass the output through a squashing element-wise nonlinearity like the sigmoid. The values passed through a sigmoid never reach the value 1 unless the input is very large so the precision produces a 1, or modifications have been made to the calculation. A model using a squashing operation on the output would likely be very sensitive to weight regularization since the model is trained to produce high-valued outputs, while at the same time trying to have low-valued weights. With a model like the PGP, which only operates on the input and learns relations, the squashing would likely lead to a recurrent diminishing of the values when predicting. Additionally, the sigmoid never reaches 0, which means that the background would successively fill up with content not present in the training set. The tanh nonlinearity solves the problem of a value never reaching 0 but still squashes the values close to 1. It is likely that the loss would not diverge as was observed when a linear mapping took place but the predictions would also be more blurry. The original model advocates linear transformations performed by U and V in Eq. (2.17), *i.e.* no bias term in Eq. (2.1). It is possible that a bias term could improve the performance of the convolutional model since it could act as a threshold ignoring blurred low-valued inputs.

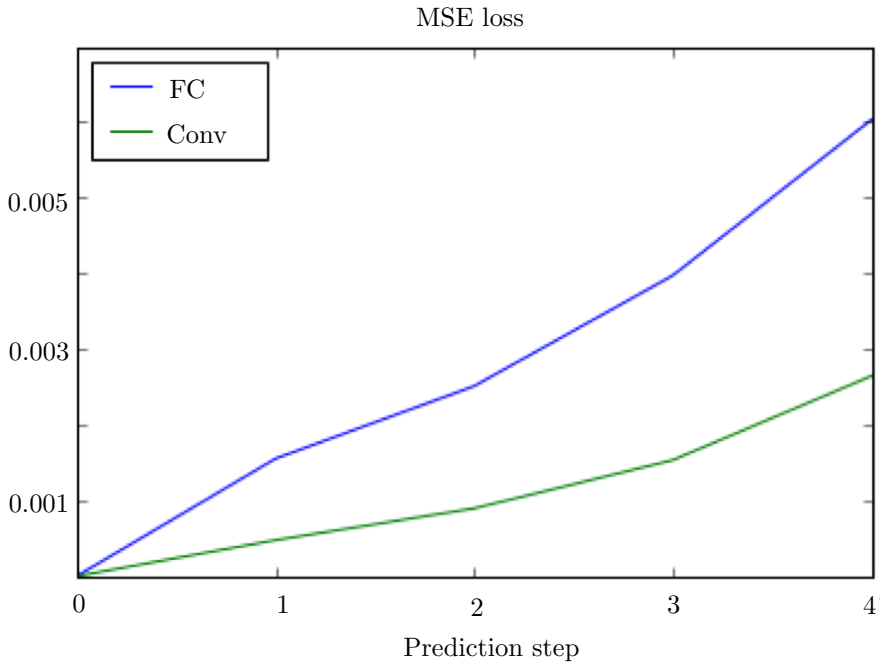


Figure 4.10: *MSE for multiple prediction steps (Eq.* (2.23)*) of the bouncing balls for a 2-layer fully-connected (FC) PGP and 2-layer convolutional PGP. Both models are trained on 2-step prediction and evaluated on 4-step prediction. The FC model consisted of (512,256) filters and (256,128) mapping units per layer respectively, with a total of 6820864 parameters to optimise. The convolutional version contained the parameters listed in Tab. 4.2, amounting to 42715 trainable parameters.*

Figure 4.11: *The sequence shows how a convolutional model predicts a ball to leave the frame when no border is added to the frames. The first three frames are seeded to the convolutional model with two layers.*

Implementing a convolutional version of the PGP required some important changes compared to the original model. The most prominent ones were the use of different activation functions, no averaging of activations in higher layers and the need for a wall around the input. All of these factors can be attributed to the local computations taking place in the convolutions. In a sparse one-dimensional test, the predictive capabilities of the convolutional model improved by increasing the size of the filters in W. When applied to the two-dimensional dataset of bouncing balls, a $W = 1 \times 1$ proved necessary in order to get interpretable filters in U, V, U^T, V^T. Due to time constraints, it was not possible to perform an explicit optimisation of parameters for the original and the convolutional version. Some qualitative results received by manual optimisation of hyperparameters showed that by using convolutions the number of weights could be reduced by a factor of 159, while still producing predictions up to four steps ahead with less than half of the loss of the original model, see Fig. 4.10.

5 Future work and conclusion

The focus of this thesis has been to extend and improve upon a previously proposed model for spatiotemporal predictions. A convolutional version of the model was successfully implemented and tested. This chapter proposes some possible improvements of the model and summarises the findings of the work.

5.1 Future work

The original PGP could manage occlusions by providing the model with a hidden state appended to the input. Extending the convolutional model in this way would lead to a belief state similar to the model proposed by Ondruska *et al.* [41]. That model was trained on non-interacting balls and was also able to predict trajectories for occluded objects. Applying the convolutional PGP in the same way is an interesting prospect and could lead to similar performance.

A drawback of the convolutional model is the computational cost. It drastically reduces the number of parameters compared to the original model but spatially operating over the whole image is computationally expensive. The computational cost can be reduced by letting the convolutional kernels only operate on every i-th pixel, $i = 1, 2, 3, \ldots$, *i.e.* using convolutions with a stride [7]. Since more transformations can occur within a selected receptive field when using a stride larger than one, more filters would likely be used by the model.

A common module used in CNNs is the max-pooling operation [9, 29]. It reduces the spatial dimension of the input, thereby reducing the computations necessary in subsequent convolutional layers. Taylor *et al.* [53] proposed a probabilistic max-pooling operation to propagate important features to the classifier in their convolutional bilinear model. The PGP is not a probabilistic model and can likely use normal max-pooling. The pooling operation is poorly conditioned to be inverted, which poses a problem when decoding a prediction to input space. Zeiler and Fergus [60] suggested an approximate way to invert a pooling operation by locally storing the location of the maximal activation. Other invertible scattering operations are mentioned by Bengio [2].

The convolutional PGP detects synchrony between a few filters at different locations of the input. This leads to multiple channels not contributing but still providing inputs to computations in later stages. Goodfellow *et al.* [10] noticed similar restrictions in normal CNNs and introduced maxout layers. Such layers can perform a pooling operation over both spatial dimensions and channels, reducing the necessary number of computations in higher layers. The aforementioned operations (stride, max-pooling and maxout) would all reduce the computational cost of the convolutional PGP, possibly at the cost of less accurate predictions. It would be interesting to study how such operations could reduce the computational cost while still maintaining accuracy.

The predictions performed by the convolutional model are limited to balls of a certain size, as seen in Section 4.2.2. The model could benefit from learning of more general transformations. One way to do this is to train the model on differently sized balls or other objects. Additionally, the relevancy of the bouncing balls dataset can be discussed. In the dataset, a model learns to capture collisions between objects. In most human environments, collisions are not so common since that is a situation we actively try to avoid. A more relevant dataset could be constructed by using differently sized balls or other objects and removing the interactions to focus more on partial occlusions. Another way to generalise the transformations would be to use low-level filters from a CNN trained on classification. The filters in a CNN become progressively more abstract as more convolution and pooling layers are stacked [29, 60]. The filters could be inserted and fixed in U in Eq. (2.17). V and V^T would then learn how to make appropriate transformations of those features due to the synchrony condition. Such low-level features are often captured by small filters but the filters in V and V^T can be of a different size to capture the transformations, as seen in Section 4.1. Using higher-layer abstractions of a CNN is tempting but due to many pooling layers, the actual position of the object might be lost and large transformations are needed to generate different activations. Without accurate positions of the features in the seeded frames, it would be difficult to make accurate predictions of position.

The mapping units in the PGP contain information about the transformations taking place between inputs. Michalski *et al.* [40] used the mapping unit activations to classify transformations between pictures. The activations in the convolutional model could also be utilised in a similar way for classification. A system combining the convolutional model with an object-classification network could be used for action recognition. The object-classifier could produce information on what is moving and the convolutional PGP would describe how it is moving. Using both streams of information, a classifier that recognises actions could be designed.

5.2 Conclusion

In this thesis, the predictive gating pyramid (PGP) proposed by Michalski *et al.* [40] has been implemented by using the deep learning framework Lasagne. The PGP consists of stacked bilinear models, known as factored gated autoencoders (GAE) [39]. A GAE detects synchrony between filters to learn transformations between inputs. The model can predict spatiotemporal data and it was tested on a dataset containing bouncing balls [51]. The PGP does not use an explicit hidden state which is often present in traditional sequential models. This offers an advantage by making the filters easier to interpret, but it cannot build an internal representation of the data to handle occlusions.

To improve the spatiotemporal abilities of the PGP, a convolutional version was implemented to learn local transformations instead of global. To better understand the convolutional model, it was first tested on a sparse one-dimensional dataset and then evaluated on the two-dimensional dataset containing bouncing balls. The convolutional model showed improved accuracy over the original by effectively halving the MSE of the predictions. Additionally, the number of parameters were reduced by a factor of 159 compared to the original model. By using convolutions, the model learned to efficiently move balls of a specific size and the filters could be used for frames of different resolutions. The original PGP required extensive training and would have to relearn the filters for frames of different resolutions. Using convolutions required fewer parameters but more computations. Some possible extensions were identified and presented to reduce the computational cost, improve the performance and to apply it to different tasks.

References

[1] Bastien, F. et al. "Theano: new features and speed improvements". *arXiv preprint arXiv:1211.5590* (2012).

[2] Bengio, Y., Courville, A., and Vincent, P. "Representation learning: A review and new perspectives". *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* **35**.8 (2013), 1798–1828.

[3] Bengio, Y. et al. "Greedy layer-wise training of deep networks". *Advances in Neural Information Processing Systems (NIPS)* **19** (2007), 153.

[4] Bottou, L. "Stochastic gradient descent tricks". *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 421–436.

[5] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". *arXiv preprint arXiv:1511.07289* (2015).

[6] Donahue, J. et al. "Long-term recurrent convolutional networks for visual recognition and description". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 2625–2634.

[7] Dumoulin, V. and Visin, F. "A guide to convolution arithmetic for deep learning". *arXiv preprint arXiv:1603.07285* (2016).

[8] Fragkiadaki, K. et al. "Recurrent network models for human dynamics". *Proceedings of the International Conference on Computer Vision (ICCV)*. IEEE. 2015, pp. 4346–4354.

[9] Goodfellow, I., Bengio, Y., and Courville, A. "Deep Learning". Book in preparation for MIT Press. 2016. URL: `http://www.deeplearningbook.org`.

[10] Goodfellow, I. et al. "Maxout networks." *Proceedings of the International Conference on Machine Learning (ICML)* **28** (2013), 1319–1327.

[11] Goodfellow, I. et al. "Generative adversarial nets". *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. 2014, pp. 2672–2680.

[12] Gregor, K. et al. "DRAW: A recurrent neural network for image generation". *arXiv preprint arXiv:1502.04623* (2015).

[13] Haykin, S. *Neural Networks: A Comprehensive Foundation*. 2nd. Prentice Hall PTR, 1998.

[14] He, K. et al. "Deep residual learning for image recognition". *arXiv preprint arXiv:1512.03385* (2015).

[15] Hebb, D. *The Organization of Behavior*. New York: Wiley & Sons, 1949.

[16] Hochreiter, S. "Untersuchungen zu dynamischen neuronalen Netzen". MA thesis. Institut für Informatik, Technische Universität, München, 1991.

[17] Hochreiter, S. and Schmidhuber, J. "Long short-term memory". *Neural computation* **9**.8 (1997), 1735–1780.

[18] Hopfield, J. "Neural networks and physical systems with emergent collective computational abilities". *Proceedings of the national academy of sciences* **79**.8 (1982), 2554–2558.

[19] Hubel, D. and Wiesel, T. "Receptive fields of single neurones in the cat's striate cortex". *The Journal of physiology* **148**.3 (1959), 574–591.

[20] Jain, A., Mao, J., and Mohiuddin, M. "Artificial neural networks: A tutorial". *IEEE computer* **29**.3 (1996), 31–44.

[21] Ji, S. et al. "3D convolutional neural networks for human action recognition". *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* **35**.1 (2013), 221–231.

[22] Kahou, S. E., Michalski, V., and Memisevic, R. "RATM: Recurrent Attentive Tracking Model". *arXiv preprint arXiv:1510.08660* (2015).

[23] Karpathy, A. et al. "Large-scale video classification with convolutional neural networks". *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (CVPR)*. 2014, pp. 1725–1732.

[24] Kingma, D. and Ba, J. "Adam: A method for stochastic optimization". *arXiv preprint arXiv:1412.6980* (2014).

[25] Konda, K., Memisevic, R., and Michalski, V. "Learning to encode motion using spatio-temporal synchrony". *International Conference on Learning Representations (ICLR)*. 2014.

[26] Konda, K. and Memisevic, R. "Learning visual odometry with a convolutional network". *International Conference on Computer Vision Theory and Applications (ICCV)*. 2015.

[27] Konda, K., Memisevic, R., and Krueger, D. "Zero-bias autoencoders and the benefits of co-adapting features". *arXiv preprint arXiv:1402.3337* (2014).

[28] Krizhevsky, A., Sutskever, I., and Hinton, G. "Imagenet classification with deep convolutional neural networks". *Advances in Neural Information Processing Systems (NIPS)*. 2012, pp. 1097–1105.

[29] LeCun, Y., Bengio, Y., and Hinton, G. "Deep learning". *Nature* **521**.7553 (2015), 436–444.

[30] LeCun, Y. et al. "Backpropagation applied to handwritten zip code recognition". *Neural computation* **1**.4 (1989), 541–551.

[31] Litman, T. "Autonomous Vehicle Implementation Predictions". *Victoria Transport Policy Institute* **28** (2014).

[32] Long, J., Shelhamer, E., and Darrell, T. "Fully convolutional networks for semantic segmentation". *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2015, pp. 3431–3440.

[33] Maas, A., Hannun, A., and Ng, A. "Rectifier nonlinearities improve neural network acoustic models". *Proceedings of the International Conference of Machine Learning (ICML)*. Vol. 30. 2013, p. 1.

[34] Mathieu, M., Couprie, C., and LeCun, Y. "Deep multi-scale video prediction beyond mean square error". *arXiv preprint arXiv:1511.05440* (2015).

[35] McCulloch, W. and Pitts, W. "A logical calculus of the ideas immanent in nervous activity". *The bulletin of mathematical biophysics* **5**.4 (1943), 115–133.

[36] Memisevic, R. "Learning to Relate Images". *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* **35**.8 (2013), 1829–1846.

[37] Memisevic, R. "Gradient-based learning of higher-order image features". *Proceedings of the International Conference on Computer Vision (ICCV)*. IEEE. 2011, pp. 1591–1598.

[38] Memisevic, R. and Hinton, G. "Unsupervised learning of image transformations". *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2007, pp. 1–8.

[39] Memisevic, R. and Hinton, G. "Learning to represent spatial transformations with factored higher-order boltzmann machines". *Neural Computation* **22**.6 (2010), 1473–1492.

[40] Michalski, V., Memisevic, R., and Konda, K. "Modeling Deep Temporal Dependencies with Recurrent Grammar Cells". *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. 2014, pp. 1925–1933.

[41] Ondruska, P. and Posner, I. "Deep Tracking: Seeing Beyond Seeing Using Recurrent Neural Networks". *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[42] Pătrăucean, V., Handa, A., and Cipolla, R. "Spatio-temporal video autoencoder with differentiable memory". *Proceedings of the International Conference on Learning Representations (ICLR)*. 2016.

[43] Pavel, M. S., Schulz, H., and Behnke, S. "Recurrent convolutional neural networks for object-class segmentation of RGB-D video". *International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2015, pp. 1–8.

[44] Ranzato, M. et al. "Video (language) modeling: a baseline for generative models of natural videos". *arXiv preprint arXiv:1412.6604* (2014).

[45] Rumelhart, D. and McClelland, J. *Parallel distributed processing*. Vol. 1. IEEE, 1988.

[46] Shah, R. and Romijnders, R. "Applying Deep Learning to Basketball Trajectories". *arXiv preprint arXiv:1608.03793* (2016).

[47] Sigaud, O. et al. "Gated networks: an inventory". *arXiv preprint arXiv:1512.03201* (2015).

[48] Srivastava, N., Mansimov, E., and Salakhutdinov, R. "Unsupervised learning of video representations using lstms". *arXiv preprint arXiv:1502.04681* **2** (2015).

[49] Srivastava, N. et al. "Dropout: A simple way to prevent neural networks from overfitting". *The Journal of Machine Learning Research* **15**.1 (2014), 1929–1958.

[50] Sutskever, I. and Hinton, G. "Learning Multilevel Distributed Representations for High-Dimensional Sequences." *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*. Vol. 2. 2007, pp. 548–555.

[51] Sutskever, I., Hinton, G., and Taylor, G. "The recurrent temporal restricted boltzmann machine". *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. 2009, pp. 1601–1608.

[52] Sutskever, I. et al. "On the importance of initialization and momentum in deep learning." *Proceedings of the International Conference on Machine Learning (ICML)* **28** (2013), 1139–1147.

[53] Taylor, G. et al. "Convolutional learning of spatio-temporal features". *Proceedings of the European Conference on Computer Vision (ECCV)*. Springer. 2010, pp. 140–153.

[54] Tieleman, T. and Hinton, G. "Lecture 6.5-rmsprop". *COURSERA: Neural networks for machine learning* (2012).

[55] Tran, D. et al. "Learning spatiotemporal features with 3d convolutional networks". *Proceedings of the International Conference on Computer Vision (ICCV)*. IEEE. 2015, pp. 4489–4497.

[56] Werbos, P. "Beyond regression: New tools for prediction and analysis in the behavioral sciences" (1974).

[57] Werbos, P. "Generalization of backpropagation with application to a recurrent gas market model". *Neural Networks* **1**.4 (1988), 339–356.

[58] Xingjian, S. et al. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". *Proceedings of Advances in Neural Information Processing Systems (NIPS)*. 2015, pp. 802–810.

[59] Yu, F. and Koltun, V. "Multi-Scale Context Aggregation by Dilated Convolutions". *Proceedings of International Conference of Learning Representations (ICLR)*. 2016.

[60] Zeiler, M. and Fergus, R. "Visualizing and understanding convolutional networks". *Proceedings of European Conference on Computer Vision (ECCV)*. Springer. 2014, pp. 818–833.