European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN), Bruges, Belgium, April 2017.

Learning Semantic Prediction using Pretrained Deep Feedforward Networks

Jörg Wagner[1,2], Volker Fischer[1], Michael Herman[1] and Sven Behnke[2]

1- Robert Bosch GmbH - 70442 Stuttgart - Germany

2- University of Bonn, Computer Science VI, Autonomous Intelligent Systems
Friedrich-Ebert-Allee 144, 53113 Bonn - Germany

Abstract. The ability to predict future environment states is crucial for anticipative behavior of autonomous agents. Deep learning based methods have proven to solve key perception challenges but currently mainly operate in a non-predictive fashion. We bridge this gap by proposing an approach to transform trained feed-forward networks into predictive ones via a combination of a recurrent predictive module with a teacher-student training strategy. This transformation can be conducted without the need of labeled data in a fully self-supervised fashion. Using simulated data, we demonstrate the ability of the resulting model to temporally predict a task-specific representation and additionally show the benefits of using our approach even when no corresponding feed-forward model is available.

1 Introduction

Deep learning based methods recently yielded impressive results in application domains such as speech recognition, computer vision, and machine translation. Especially in visual perception, deep convolutional neural networks dominate the majority of current benchmarks. Due to their ability to model complex data sets as well as to generalize to unseen examples, they have the potential to address key challenges of autonomous systems. Among these are the semantic perception of their environment as well as the prediction of future environment states in order to enable acting in an anticipatory way.

Current deep learning-based approaches which predict the future state of the environment either require a large set of labeled sequences [1], use additional post-processing steps to perform the prediction [2], or predict the raw sensory data [3, 4, 5]. The last category is very popular in recent research, due to the fact that models can be trained without access to labels. These approaches make use of the sequential nature of sensor streams and directly predict the next frame in a sequence of measurements given the history of previous measurements. They have been used to predict the next image in a video [3], to predict radar measurements [4], and to predict and filter future laser scans [5].

In most applications, one is more interested in predicting an application-specific representation of the world rather than the next measurement. To handle such cases without the need for additional training data, one could, for example, use an existing feed-forward network and apply it to the output of a model which predicts raw data. Such an approach in general requires many parameters and has a high computational overhead. This is due to the fact that the predictive model has to reconstruct the whole measurement of the next time step, which includes additional information not required to make task-specific predictions. If a limited amount of labeled data is available, one could alternatively learn a general-purpose predictive feature extractor and train a task-specific subnetwork on top of it [6, 7]. Besides the disadvantage of needing labeled data, the learned feature representation may not necessarily be suitable for solving the task.

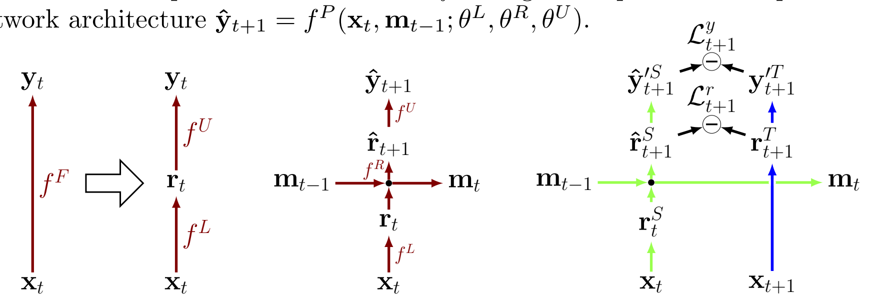
To address these issues, we exploit the availability of numerous non-predictive networks and propose an approach to transform these networks into predictive ones. Our approach does not need labeled sequences, is trainable in an end-to-end fashion and can be used to generate a predictive model for any task-specific representation. We additionally demonstrate that our approach is beneficial in cases when no feed-forward network is available and one is tasked with training a predictive model based on a limited amount of labeled data.

2 Predictive Transformation

To convert a regular feed-forward network into a predictive one, we first transform it into a recurrent network by introducing an additional network module (Sections 2.1 and 2.2). The weights of the newly generated predictive network are trained using a teacher-student-like [8] approach (Section 2.3).

2.1 Predictive Network Architecture

We assume that a trained feed-forward network $\mathbf{y}_t = f^F(\mathbf{x}_t; \theta^F)$ is given, which receives a measurement $\mathbf{x}_t$ and generates a corresponding application specific target $\mathbf{y}_t$. To transform this model into a predictive one, we first split it into two parts (Fig. 1a). The lower network part $\mathbf{r}_t = f^L(\mathbf{x}_t; \theta^L)$ generates an abstract task-specific representation $\mathbf{r}_t$ and the upper part $\mathbf{y}_t = f^U(\mathbf{r}_t; \theta^U)$ computes the target $\mathbf{y}_t$. The split of the network has to be chosen task-specifically. The representation $\mathbf{r}_t$ is predicted into the future using a recurrent predictive module $\hat{\mathbf{r}}_{t+1} = f^R(\mathbf{r}_t, \mathbf{m}_{t-1}; \theta^R)$. This module predicts the expected representation of the next time step $\hat{\mathbf{r}}_{t+1}$ based on the current representation $\mathbf{r}_t$ as well as a hidden state $\mathbf{m}_{t-1}$. To generate the target $\hat{\mathbf{y}}_{t+1} = f^U(\hat{\mathbf{r}}_{t+1}; \theta^U)$, the predicted representation can be passed to the network f^U. Fig. 1b depicts the full predictive network architecture $\hat{\mathbf{y}}_{t+1} = f^P(\mathbf{x}_t, \mathbf{m}_{t-1}; \theta^L, \theta^R, \theta^U)$.



(a) Feed-forward model. (b) Predictive model. (c) Teacher-student structure.

Fig. 1: Predictive network architecture and teacher-student training structure.

2.2 Recurrent Predictive Module

The recurrent predictive module is tasked with predicting a representation of a feed-forward network into the future. Thus, it has to learn the dynamics of the representation based on a sequence of previous representations and predict the expected next state. A natural model choice to implement such a behavior are Recurrent Neural Networks (RNNs). Long Short Term Memory (LSTM) models as a variant of RNNs have proven to produce state-of-the-art results in various sequence modeling tasks. Due to the spatial and local nature of the data in our experiments, we use a convolutional LSTM as suggested by Shi et al. [4]. The recurrent part of the predictive module is defined by:

$$\begin{aligned}
\mathbf{k}_t &= \sigma(\mathbf{W}_{rk} * \mathbf{r}_t + \mathbf{W}_{hk} * \mathbf{h}_{t-1} + \mathbf{W}_{ck} \circ \mathbf{c}_{t-1} + \mathbf{b}_k),\ \forall \mathbf{k} \in \{\mathbf{i}, \mathbf{f}\}\,; && (1)\\
\mathbf{c}_t &= \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tanh(\mathbf{W}_{rc} * \mathbf{r}_t + \mathbf{W}_{hc} * \mathbf{h}_{t-1} + \mathbf{b}_c); && (2)\\
\mathbf{o}_t &= \sigma(\mathbf{W}_{ro} * \mathbf{r}_t + \mathbf{W}_{ho} * \mathbf{h}_{t-1} + \mathbf{W}_{co} \circ \mathbf{c}_t + \mathbf{b}_o); && (3)\\
\mathbf{h}_t &= \mathbf{o}_t \circ \tanh(\mathbf{c}_t); && (4)
\end{aligned}$$

where $*$ is the convolutional operator and $\circ$ the Hadamard product. For brevity, we will summarize $\mathbf{c}_t$ and $\mathbf{h}_t$ by $\mathbf{m}_t$. To obtain a predicted representation $\hat{\mathbf{r}}_{t+1}$, we pass the cell state $\mathbf{c}_t$ through a convolutional layer $\hat{\mathbf{r}}_{t+1} = \phi(\mathbf{W}_{cr} * \mathbf{c}_t + \mathbf{b}_{cr}) = f^R(\mathbf{r}_t, \mathbf{m}_{t-1}; \theta^R)$. The parameter θ^R represents all trainable weights of the predictive module f^R and ϕ denotes the nonlinearity. The weight tensor $\mathbf{W}_{cr}$ has to be chosen in such a way that the predicted representation $\hat{\mathbf{r}}_{t+1}$ and the input representation $\mathbf{r}_t$ have equal dimensions. In cases when the target task is more complex, one could obtain a deeper module by stacking multiple LSTMs.

2.3 Predictive Knowledge Transfer

We use the teacher-student paradigm [8] to train the predictive network architecture in a self-supervised manner. This paradigm is usually employed to compress a well trained deep network into a smaller network by using the output of the teacher as a supervision signal. The feed-forward network f^F serves as the teacher in our setting. To generate a predictive supervision signal, we provide the teacher with the measurement of the next time step and force the student network to mimic the teacher given the current measurement as well as the internal memory (Fig. 1c). The trained weights of the teacher, which we will denote as $\theta_T^F = \{\theta_T^L, \theta_T^U\}$ are fixed during the training procedure. The weights of the student $\theta_S^P = \{\theta_S^L, \theta_S^R, \theta_S^U\}$ are optimized using the supervision signal. In addition, we initialize all weights of the student which are not part of the predictive module with the corresponding weights of the teacher. Since the initial weights θ_S^L and θ_S^U are thus already well trained, one could also fix them and only train the weights θ_S^R. We evaluate both options in the experiments. To compute the gradient with respect to the weights, we use Backpropagation Through Time (BPTT) and unroll the recurrent network for N time steps.

We evaluate two losses to train the student: a loss $\mathcal{L}^r$ defined on representation level and a loss $\mathcal{L}^y$ defined on the network outputs. The loss $\mathcal{L}^r$ enforces a similarity between the predicted representation $\hat{\mathbf{r}}_{t+1}^{S}$ of the student and the representation $\mathbf{r}_{t+1}^{T}$ of the teacher for the last N steps:

$$\mathcal{L}^{r}(\theta_S^L, \theta_S^R) = \sum\nolimits_{t=1}^{N} \lambda_t \tfrac{1}{2} \left\| f^R(f^L(\mathbf{x}_t; \theta_S^L), \mathbf{m}_{t-1}; \theta_S^R) - f^L(\mathbf{x}_{t+1}; \theta_T^L) \right\|_2^2, \tag{5}$$

where λ_t denotes a time-dependent weighting factor. The loss $\mathcal{L}^y$ on the other hand enforces a similarity between the task-specific output of the two networks. Due to the fact that the task of our experiments is a classification task, we will hereinafter focus on a classification-specific output loss $\mathcal{L}^y$:

$$\mathcal{L}^{y}(\theta_S^P) = \sum\nolimits_{t=1}^{N} \lambda_t \mathcal{H}\Big(f^P(\mathbf{x}_t, \mathbf{m}_{t-1}; \theta_S^P, \tau), f^F(\mathbf{x}_{t+1}; \theta_T^F, \tau) \Big), \tag{6}$$

where $\mathcal{H}$ denotes the cross entropy and $\hat{\mathbf{y}}_{t+1}^{\prime S} = f^P(\mathbf{x}_t, \mathbf{m}_{t-1}; \theta_S^P, \tau)$ as well as $\mathbf{y}_{t+1}^{\prime T} = f^F(\mathbf{x}_{t+1}; \theta_T^F, \tau)$ represent softened versions of the respective outputs computed by using a softmax temperature $\tau > 1$. Training with softened outputs is in general beneficial as argued by Hinton et al. [8].

3 Experiments

To systematically evaluate different aspects of the proposed approach, we use a simulated video dataset. The data emulates a 2D environment of 64×64 pixels, in which rectangles represent walls and environment borders, circles represent moving objects and squares represent static foreground objects (Fig. 2a). The circles elastically collide with other circles, walls, and borders. Squares occlude all other objects as well as each other. Color and size of the objects, the number of walls, squares and circles as well as the velocity of circles are randomly sampled for each sequence. Additionally, we add independent Gaussian noise with zero mean and a variance of 0.005 to each pixel. In total, our dataset contains 10,000 sequences of length 16, which are split into 6,000 training sequences and 2,000 validation and test sequences, respectively. As a task we chose pixel-level semantic segmentation [9, 10] with four classes: background, walls and borders, circles, and squares. A label is only available for the last image in each sequence. In addition, we assume that only 1,000 sequences of the training data are labeled.

The used feed-forward model ($\mathcal{FF}$) is given by the following shortcut notation $C'(32,3)$-$C'(32,3)$-P-$C'(64,3)$-$C'(64,3)$-P-$C'(64,3)$-D-$C'(512,1)$-D-$C(4,1)$-UC where $C(d,f)$ is a convolutional layer with d filters, a filter size of $f \times f$ and a stride of 1, P a pooling layer with non-overlapping 2×2 regions, D a dropout layer, and UC a deconvolutional layer. The deconvolutional layer upsamples the target to have the same dimensions as the input image. All convolutional layers except the last one use a leaky ReLU with a leakiness of 0.01. The last convolutional layer is linear and the deconvolutional layer is followed by a pixel-wise softmax function. A prime additionally marks layers which apply batch normalization. We train the feed-forward model using the 1,000 labeled images of the training data and a per-pixel multinomial logistic loss. The achieved mean intersection-over-union (IoU) [10] on the test data is 89.02 %.

The predictive network is constructed in accordance with Sec. 2.1, by splitting $\mathcal{FF}$ beneath the first dropout layer. The parameters of the predictive module are listed in Table 1. We additionally apply dropout to activations entering the LSTM and use zoneout within the LSTM. The LSTM as well as the output convolution of the predictive module are followed by a batch normalization layer.

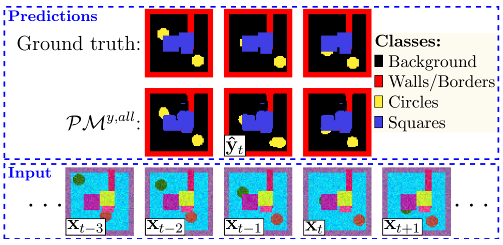
	Filter size	Num. filters
$\mathbf{W}_{r*}$	5×5	128
$\mathbf{W}_{h*}$	7×7	128
$\mathbf{W}_{cr}$	3×3	64
ϕ	leaky ReLU, $\alpha = 0.01$	

Table 1: Predictive module parameters.

We train the predictive model four times using the different knowledge transfer variations of Sec. 2.3. Versions $\mathcal{PM}^{r,all}$ and $\mathcal{PM}^{y,all}$, respectively, use the representation and output loss and optimize all model parameters. Version $\mathcal{PM}^{r,rec}$ and $\mathcal{PM}^{y,rec}$ are trained by only optimizing the parameters θ_S^R. The training is conducted utilizing all 6,000 training sequences without labels. For each sequence, we provide the first 15 images to the student and the teacher receives the last 15 images. The weighting factor is set to $\lambda_t = ((t-1)/(N-1))^5$.

These models are compared with two baselines. Baseline $\mathcal{PM}^{copy}$ uses the weights of $\mathcal{FF}$ and a copy function ($\hat{\mathbf{r}}_{t+1} = \mathbf{r}_t$). Baseline $\mathcal{PM}^{sup}$ is a predictive model trained from scratch using only the 1,000 labeled training sequences. In this setting, the predictive model receives the first 15 images of each sequence and predicts labels of the 16th image. Additionally, we train a predictive model $\mathcal{PM}^{pre}$ which uses the weights of the best self-supervised model $\mathcal{PM}^{y,all}$ as an initialization and then fine-tunes all weights in accordance to $\mathcal{PM}^{sup}$. To make the comparison of the different models as fair as possible, we perform an extensive random search to determine training and regularization parameters.

We summarize the results of our experiments in Fig. 2 by reporting the mean IoU on test data as well as visualizing example predictions of $\mathcal{PM}^{y,all}$. All models trained with our approach significantly outperform the baseline $\mathcal{PM}^{copy}$. The best model $\mathcal{PM}^{y,all}$ improves the mean IoU by more than 7 %, compared to $\mathcal{PM}^{copy}$. This implies that the recurrent predictive module successfully learns a dynamic model. Predictive capabilities are additionally visible in Fig. 2a. The model is able to recognize different object types and to predict the future pixel-wise semantic labels. It can even resolve heavy occlusion of one of the circles in Image $\mathbf{x}_t$. The predictions are rather coarse which is due to the structure of the chosen $\mathcal{FF}$. When using the loss $\mathcal{L}^r$, it is beneficial to only train parameters of the recurrent predictive module. For the loss $\mathcal{L}^y$, the best results can be



(a) Example predictions of model $\mathcal{PM}^{y,all}$.

Model	**Mean IoU%**
$\boldsymbol{\mathcal{PM}^{y,all}}$	**82.97 %**
$\mathcal{PM}^{r,all}$	79.63 %
$\mathcal{PM}^{y,rec}$	80.05 %
$\mathcal{PM}^{r,rec}$	80.59 %
$\mathcal{PM}^{copy}$	75.72 %
$\mathcal{PM}^{sup}$	80.63 %
$\mathcal{PM}^{pre}$	84.96 %

(b) Mean IoU on test data.

Fig. 2: Results of the predictive pixel-wise semantic segmentation experiments.

obtained when all weights are optimized. The best model trained on the output loss $\mathcal{L}^y$ outperformed the best model trained on the representation loss $\mathcal{L}^r$.

For $\mathcal{PM}^{sup}$, we observed a mean IoU of 80.63 % which is less than the result of our best model $\mathcal{PM}^{y,all}$. This is most likely due to the benefits of training with softened outputs as well as due to the additional information from unlabeled sequences. The overall best results can be achieved when the model $\mathcal{PM}^{y,all}$ is further fine-tuned using labeled training data (see $\mathcal{PM}^{pre}$). These results suggest that our approach is beneficial when training a predictive model without access to a corresponding feed-forward model. One could then employ a three-step training approach consisting of: Training of a corresponding feed-forward model using labeled sequences, predictive transformation in accordance to Sec. 2 using additional unlabeled sequences, and fine-tuning using labeled data.

4 Conclusion

We proposed an approach to transform a given, trained feed-forward network into a predictive network by introducing a recurrent predictive module. To optimize the weights of the resulting model, we propose a teacher-student-like training approach, which can be conducted without access to labels. Our analysis on simulated motion sequences shows that the resulting network can model the dynamics of a representation and thereby predict the task-specific target for the next time step. We additionally demonstrate the advantages of using our approach when training a predictive model on a limited amount of labeled data without the availability of a corresponding feed-forward model.

References

[1] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese. Social LSTM: Human trajectory prediction in crowded spaces. In *CVPR*, 2016.

[2] D. Ribeiro, A. Mateus, C. Nascimento J. and P. Miraldo. A real-time pedestrian detector using deep learning for human-aware navigation. *arXiv preprint arXiv:1607.04441*, 2016.

[3] A. Ranzato M. A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra. Video (language) modeling: A baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014.

[4] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-C. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *NIPS*, 2015.

[5] P. Ondruska and I. Posner. Deep tracking: Seeing beyond seeing using recurrent neural networks. In *The Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[6] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using LSTMs. In *ICML*, 2015.

[7] C. Vondrick, H. Pirsiavash, and A. Torralba. Anticipating visual representations from unlabeled video. In *CVPR*, 2016.

[8] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In *Deep Learning and Representation Learning Workshop, NIPS*, 2014.

[9] M. S. Pavel, H. Schulz, and S. Behnke. Object class segmentation of RGB-D video using recurrent convolutional neural networks. *Neural Networks, Elsevier*, 2017.

[10] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.