Action-Conditional Video Prediction using Deep Networks in Atari Games

Junhyuk Oh **Xiaoxiao Guo** **Honglak Lee** **Richard Lewis** **Satinder Singh**
University of Michigan, Ann Arbor, MI 48109, USA
{junhyuk,guoxiao,honglak,rickl,baveja}@umich.edu

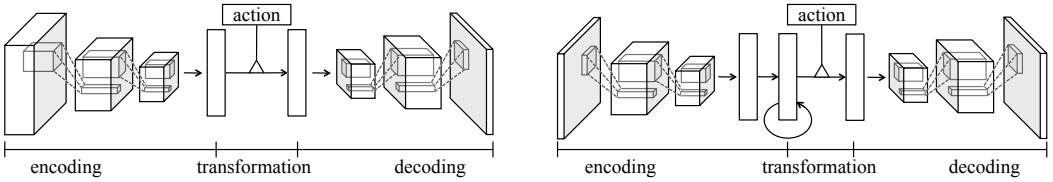
Abstract

Motivated by vision-based reinforcement learning (RL) problems, in particular Atari games from the recent benchmark Aracade Learning Environment (ALE), we consider spatio-temporal prediction problems where future image-frames depend on control variables or actions as well as previous frames. While not composed of natural scenes, frames in Atari games are high-dimensional in size, can involve tens of objects with one or more objects being controlled by the actions directly and many other objects being influenced indirectly, can involve entry and departure of objects, and can involve deep partial observability. We propose and evaluate two deep neural network architectures that consist of encoding, action-conditional transformation, and decoding layers based on convolutional neural networks and recurrent neural networks. Experimental results show that the proposed architectures are able to generate visually-realistic frames that are also useful for control over approximately 100-step action-conditional futures in some games. To the best of our knowledge, this paper is the first to make and evaluate long-term predictions on high-dimensional video conditioned by control inputs.

1 Introduction

Over the years, deep learning approaches (see [5, 26] for survey) have shown great success in many visual perception problems (e.g., [16, 7, 32, 9]). However, modeling videos (building a generative model) is still a very challenging problem because it often involves high-dimensional natural-scene data with complex temporal dynamics. Thus, recent studies have mostly focused on modeling simple video data, such as bouncing balls or small patches, where the next frame is highly-predictable given the previous frames [29, 20, 19]. In many applications, however, future frames depend not only on previous frames but also on control or action variables. For example, the first-person-view in a vehicle is affected by wheel-steering and acceleration. The camera observation of a robot is similarly dependent on its movement and changes of its camera angle. More generally, in vision-based reinforcement learning (RL) problems, learning to predict future images conditioned on actions amounts to learning a model of the dynamics of the agent-environment interaction, an essential component of model-based approaches to RL. In this paper, we focus on Atari games from the Arcade Learning Environment (ALE) [1] as a source of challenging action-conditional video modeling problems. While not composed of natural scenes, frames in Atari games are high-dimensional, can involve tens of objects with one or more objects being controlled by the actions directly and many other objects being influenced indirectly, can involve entry and departure of objects, and can involve deep partial observability. To the best of our knowledge, this paper is the first to make and evaluate long-term predictions on high-dimensional images conditioned by control inputs.

This paper proposes, evaluates, and contrasts two spatio-temporal prediction architectures based on deep networks that incorporate action variables (See Figure 1). Our experimental results show that our architectures are able to generate realistic frames over 100-step action-conditional future frames without diverging in some Atari games. We show that the representations learned by our architectures 1) approximately capture natural similarity among actions, and 2) discover which objects are directly controlled by the agent's actions and which are only indirectly influenced or not controlled. We evaluated the usefulness of our architectures for control in two ways: 1) by replacing emulator frames with predicted frames in a previously-learned model-free controller (DQN; DeepMind's state



(a) Feedforward encoding (b) Recurrent encoding

Figure 1: Proposed Encoding-Transformation-Decoding network architectures.

of the art Deep-Q-Network for Atari Games [21]), and 2) by using the predicted frames to drive a more informed than random exploration strategy to improve a model-free controller (also DQN).

2 Related Work

Video Prediction using Deep Networks. The problem of video prediction has led to a variety of architectures in deep learning. A recurrent temporal restricted Boltzmann machine (RTRBM) [29] was proposed to learn temporal correlations from sequential data by introducing recurrent connections in RBM. A structured RTRBM (sRTRBM) [20] scaled up RTRBM by learning dependency structures between observations and hidden variables from data. More recently, Michalski et al. [19] proposed a higher-order gated autoencoder that defines multiplicative interactions between consecutive frames and mapping units, and showed that temporal prediction problem can be viewed as learning and inferring higher-order interactions between consecutive images. Srivastava et al. [28] applied a sequence-to-sequence learning framework [31] to a video domain, and showed that long short-term memory (LSTM) [12] networks are capable of generating video of bouncing handwritten digits. In contrast to these previous studies, this paper tackles problems where control variables affect temporal dynamics, and in addition scales up spatio-temporal prediction to larger-size images.

ALE: Combining Deep Learning and RL. Atari 2600 games provide challenging environments for RL because of high-dimensional visual observations, partial observability, and delayed rewards. Approaches that combine deep learning and RL have made significant advances [21, 22, 11]. Specifically, DQN [21] combined Q-learning [36] with a convolutional neural network (CNN) and achieved state-of-the-art performance on many Atari games. Guo et al. [11] used the ALE-emulator for making action-conditional predictions with slow UCT [15], a Monte-Carlo tree search method, to generate training data for a fast-acting CNN, which outperformed DQN on several domains. Throughout this paper we will use DQN to refer to the architecture used in [21] (a more recent work [22] used a deeper CNN with more data to produce the currently best-performing Atari game players).

Action-Conditional Predictive Model for RL. The idea of building a predictive model for vision-based RL problems was introduced by Schmidhuber and Huber [27]. They proposed a neural network that predicts the attention region given the previous frame and an *attention-guiding* action. More recently, Lenz et al. [17] proposed a recurrent neural network with multiplicative interactions that predicts the physical coordinate of a robot. Compared to this previous work, our work is evaluated on much higher-dimensional data with complex dependencies among observations. There have been a few attempts to learn from ALE data a transition-model that makes predictions of future frames. One line of work [3, 4] divides game images into patches and applies a Bayesian framework to predict patch-based observations. However, this approach assumes that neighboring patches are enough to predict the center patch, which is not true in Atari games because of many complex interactions. The evaluation in this prior work is 1-step prediction loss; in contrast, here we make and evaluate long-term predictions both for quality of pixels generated and for usefulness to control.

3 Proposed Architectures and Training Method

The goal of our architectures is to learn a function $f : \mathbf{x}_{1:t}, \mathbf{a}_t \rightarrow \mathbf{x}_{t+1}$, where $\mathbf{x}_t$ and $\mathbf{a}_t$ are the frame and action variables at time t, and $\mathbf{x}_{1:t}$ are the frames from time 1 to time t. Figure 1 shows our two architectures that are each composed of encoding layers that extract spatio-temporal features from the input frames (§3.1), action-conditional transformation layers that transform the encoded features into a prediction of the next frame in high-level feature space by introducing action variables as additional input (§3.2) and finally decoding layers that map the predicted high-level features into pixels (§3.3). Our contributions are in the novel action-conditional deep convolutional architectures for high-dimensional, long-term prediction as well as in the novel use of the architectures in vision-based RL domains.

3.1 Two Variants: Feedforward Encoding and Recurrent Encoding

Feedforward encoding takes a fixed history of previous frames as an input, which is concatenated through channels (Figure 1a), and stacked convolution layers extract spatio-temporal features directly from the concatenated frames. The encoded feature vector $\mathbf{h}_t^{enc} \in \mathbb{R}^n$ at time t is:

$$\mathbf{h}_t^{enc} = \text{CNN}\left(\mathbf{x}_{t-m+1:t}\right), \tag{1}$$

where $\mathbf{x}_{t-m+1:t} \in \mathbb{R}^{(m\times c)\times h\times w}$ denotes m frames of $h \times w$ pixel images with c color channels. CNN is a mapping from raw pixels to a high-level feature vector using multiple convolution layers and a fully-connected layer at the end, each of which is followed by a non-linearity. This encoding can be viewed as *early-fusion* [14] (other types of fusions, e.g., *late-fusion* or 3D convolution [35] can also be applied to this architecture).

Recurrent encoding takes one frame as an input for each time-step and extracts spatio-temporal features using an RNN in which the temporal dynamics is modeled by the recurrent layer on top of the high-level feature vector extracted by convolution layers (Figure 1b). In this paper, LSTM without peephole connection is used for the recurrent layer as follows:

$$[\mathbf{h}_t^{enc}, \mathbf{c}_t] = \text{LSTM}\left(\text{CNN}\left(\mathbf{x}_t\right), \mathbf{h}_{t-1}^{enc}, \mathbf{c}_{t-1}\right), \tag{2}$$

where $\mathbf{c}_t \in \mathbb{R}^n$ is a *memory cell* that retains information from a deep history of inputs. Intuitively, $\text{CNN}\left(\mathbf{x}_t\right)$ is given as input to the LSTM so that the LSTM captures temporal correlations from high-level spatial features.

3.2 Multiplicative Action-Conditional Transformation

We use multiplicative interactions between the encoded feature vector and the control variables:

$$h_{t,i}^{dec} = \sum_{j,l} W_{ijl} h_{t,j}^{enc} a_{t,l} + b_i, \tag{3}$$

where $\mathbf{h}_t^{enc} \in \mathbb{R}^n$ is an encoded feature, $\mathbf{h}_t^{dec} \in \mathbb{R}^n$ is an action-transformed feature, $\mathbf{a}_t \in \mathbb{R}^a$ is the action-vector at time t, $\mathbf{W} \in \mathbb{R}^{n\times n\times a}$ is 3-way tensor weight, and $\mathbf{b} \in \mathbb{R}^n$ is bias. When the action $\mathbf{a}$ is represented using one-hot vector, using a 3-way tensor is equivalent to using different weight matrices for each action. This enables the architecture to model different transformations for different actions. The advantages of multiplicative interactions have been explored in image and text processing [33, 30, 18]. In practice the 3-way tensor is not scalable because of its large number of parameters. Thus, we approximate the tensor by factorizing into three matrices as follows [33]:

$$\mathbf{h}_t^{dec} = \mathbf{W}^{dec}\left(\mathbf{W}^{enc}\mathbf{h}_t^{enc} \odot \mathbf{W}^a \mathbf{a}_t\right) + \mathbf{b}, \tag{4}$$

where $\mathbf{W}^{dec} \in \mathbb{R}^{n\times f}, \mathbf{W}^{enc} \in \mathbb{R}^{f\times n}, \mathbf{W}^a \in \mathbb{R}^{f\times a}, \mathbf{b} \in \mathbb{R}^n$, and f is the number of factors. Unlike the 3-way tensor, the above factorization shares the weights between different actions by mapping them to the size-f factors. This sharing may be desirable relative to the 3-way tensor when there are common temporal dynamics in the data across different actions (discussed further in §4.3).

3.3 Convolutional Decoding

It has been recently shown that a CNN is capable of generating an image effectively using upsampling followed by convolution with stride of 1 [8]. Similarly, we use the "inverse" operation of convolution, called deconvolution, which maps 1×1 spatial region of the input to $d \times d$ using deconvolution kernels. The effect of $s \times s$ upsampling can be achieved without explicitly upsampling the feature map by using stride of s. We found that this operation is more efficient than upsampling followed by convolution because of the smaller number of convolutions with larger stride.

In the proposed architecture, the transformed feature vector $\mathbf{h}^{dec}$ is decoded into pixels as follows:

$$\hat{\mathbf{x}}_{t+1} = \text{Deconv}\left(\text{Reshape}\left(\mathbf{h}^{dec}\right)\right), \tag{5}$$

where Reshape is a fully-connected layer where hidden units form a 3D feature map, and Deconv consists of multiple deconvolution layers, each of which is followed by a non-linearity except for the last deconvolution layer.

3.4 Curriculum Learning with Multi-Step Prediction

It is almost inevitable for a predictive model to make noisy predictions of high-dimensional images. When the model is trained on a 1-step prediction objective, small prediction errors can compound

through time. To alleviate this effect, we use a multi-step prediction objective. More specifically, given the training data $D = \left\{ \left(\left(\mathbf{x}_1^{(i)}, \mathbf{a}_1^{(i)} \right), ..., \left(\mathbf{x}_{T_i}^{(i)}, \mathbf{a}_{T_i}^{(i)} \right) \right) \right\}_{i=1}^{N}$, the model is trained to minimize the average squared error over K-step predictions as follows:

$$\mathcal{L}_K(\theta) = \frac{1}{2K} \sum_i \sum_t \sum_{k=1}^{K} \left\| \hat{\mathbf{x}}_{t+k}^{(i)} - \mathbf{x}_{t+k}^{(i)} \right\|^2, \tag{6}$$

where $\hat{\mathbf{x}}_{t+k}^{(i)}$ is a k-step future prediction. Intuitively, the network is repeatedly *unrolled* through K time steps by using its prediction as an input for the next time-step.

The model is trained in multiple phases based on increasing K as suggested by Michalski et al. [19]. In other words, the model is trained to predict short-term future frames and fine-tuned to predict longer-term future frames after the previous phase converges. We found that this curriculum learning [6] approach is necessary to stabilize the training. A stochastic gradient descent with backpropagation through time (BPTT) is used to optimize the parameters of the network.

4 Experiments

In the experiments that follow, we have the following goals for our two architectures. 1) To evaluate the predicted frames in two ways: qualitatively evaluating the generated video, and quantitatively evaluating the pixel-based squared error, 2) To evaluate the usefulness of predicted frames for control in two ways: by replacing the emulator's frames with predicted frames for use by DQN, and by using the predictions to improve exploration in DQN, and 3) To analyze the representations learned by our architectures. We begin by describing the details of the data, and model architecture, and baselines.

Data and Preprocessing. We used our replication of DQN to generate game-play video datasets using an ϵ-greedy policy with $\epsilon = 0.3$, i.e. DQN is forced to choose a random action with 30% probability. For each game, the dataset consists of about $500,000$ training frames and $50,000$ test frames with actions chosen by DQN. Following DQN, actions are chosen once every 4 frames which reduces the video from 60fps to 15fps. The number of actions available in games varies from 3 to 18, and they are represented as one-hot vectors. We used full-resolution RGB images (210×160) and preprocessed the images by subtracting mean pixel values and dividing each pixel value by 255.

Network Architecture. Across all game domains, we use the same network architecture as follows. The encoding layers consist of 4 convolution layers and one fully-connected layer with 2048 hidden units. The convolution layers use 64 (8×8), 128 (6×6), 128 (6×6), and 128 (4×4) filters with stride of 2. Every layer is followed by a rectified linear function [23]. In the recurrent encoding network, an LSTM layer with 2048 hidden units is added on top of the fully-connected layer. The number of factors in the transformation layer is 2048. The decoding layers consists of one fully-connected layer with 11264 $(= 128 \times 11 \times 8)$ hidden units followed by 4 deconvolution layers. The deconvolution layers use 128 (4×4), 128 (6×6), 128 (6×6), and 3 (8×8) filters with stride of 2. For the feedforward encoding network, the last 4 frames are given as an input for each time-step. The recurrent encoding network takes one frame for each time-step, but it is unrolled through the last 11 frames to initialize the LSTM hidden units before making a prediction. Our implementation is based on Caffe toolbox [13].

Details of Training. We use the curriculum learning scheme above with three phases of increasing prediction step objectives of 1, 3 and 5 steps, and learning rates of 10^{-4}, 10^{-5}, and 10^{-5}, respectively. RMSProp [34, 10] is used with momentum of 0.9, (squared) gradient momentum of 0.95, and min squared gradient of 0.01. The batch size for each training phase is 32, 8, and 8 for the feedforward encoding network and 4, 4, and 4 for the recurrent encoding network, respectively. When the recurrent encoding network is trained on 1-step prediction objective, the network is unrolled through 20 steps and predicts the last 10 frames by taking ground-truth images as input. Gradients are clipped at $[-0.1, 0.1]$ before non-linearity of each gate of LSTM as suggested by [10].

Two Baselines for Comparison. The first baseline is a multi-layer perceptron (*MLP*) that takes the last frame as input and has 4 hidden layers with 400, 2048, 2048, and 400 units. The action input is concatenated to the second hidden layer. This baseline uses approximately the same number of parameters as the recurrent encoding model. The second baseline, no-action feedforward (or *naFf*), is the same as the feedforward encoding model (Figure 1a) except that the transformation layer consists of one fully-connected layer that does not get the action as input.

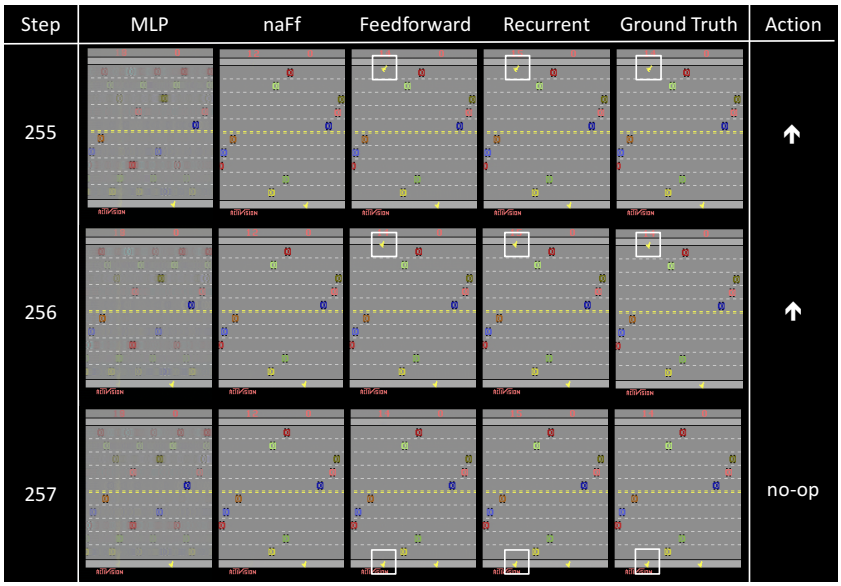
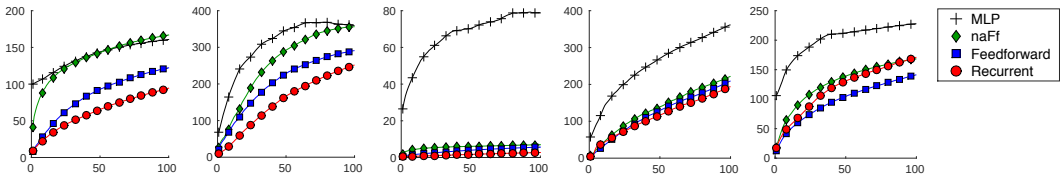


Figure 2: Example of predictions over 250 steps in Freeway. The 'Step' and 'Action' columns show the number of prediction steps and the actions taken respectively. The white boxes indicate the object controlled by the agent. From prediction step 256 to 257 the controlled object crosses the top boundary and reappears at the bottom; this non-linear shift is predicted by our architectures and is not predicted by MLP and naFf. The horizontal movements of the uncontrolled objects are predicted by our architectures and naFf but not by MLP.



(a) Seaquest (b) Space Invaders (c) Freeway (d) QBert (e) Ms Pacman

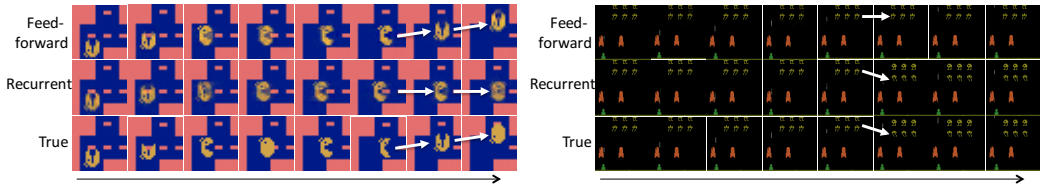
Figure 3: Mean squared error over 100-step predictions

4.1 Evaluation of Predicted Frames

Qualitative Evaluation: Prediction video. The prediction videos of our models and baselines are available in the supplementary material and at the following website: https://sites.google.com/a/umich.edu/junhyuk-oh/action-conditional-video-prediction. As seen in the videos, the proposed models make qualitatively reasonable predictions over 30–500 steps depending on the game. In all games, the MLP baseline quickly diverges, and the naFf baseline fails to predict the controlled object. An example of long-term predictions is illustrated in Figure 2. We observed that both of our models predict complex local translations well such as the movement of vehicles and the controlled object. They can predict interactions between objects such as collision of two objects. Since our architectures effectively extract hierarchical features using CNN, they are able to make a prediction that requires a global context. For example, in Figure 2, the model predicts the sudden change of the location of the controlled object (from the top to the bottom) at 257-step.

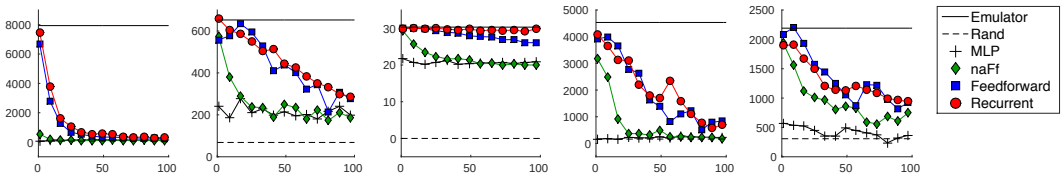
However, both of our models have difficulty in accurately predicting small objects, such as bullets in Space Invaders. The reason is that the squared error signal is small when the model fails to predict small objects during training. Another difficulty is in handling stochasticity. In Seaquest, e.g., new objects appear from the left side or right side randomly, and so are hard to predict. Although our models do generate new objects with reasonable shapes and movements (e.g., after appearing they move as in the true frames), the generated frames do not necessarily match the ground-truth.

Quantitative Evaluation: Squared Prediction Error. Mean squared error over 100-step predictions is reported in Figure 3. Our predictive models outperform the two baselines for all domains. However, the gap between our predictive models and naFf baseline is not large except for Seaquest. This is due to the fact that the object controlled by the action occupies only a small part of the image.



(a) Ms Pacman (28 × 28 cropped) (b) Space Invaders (90 × 90 cropped)

Figure 4: Comparison between two encoding models (feedforward and recurrent). (a) Controlled object is moving along a horizontal corridor. As the recurrent encoding model makes a small translation error at 4th frame, the true position of the object is in the crossroad while the predicted position is still in the corridor. The (true) object then moves upward which is not possible in the predicted position and so the predicted object keeps moving right. This is less likely to happen in feedforward encoding because its position prediction is more accurate. (b) The objects move down after staying at the same location for the first five steps. The feedforward encoding model fails to predict this movement because it only gets the last four frames as input, while the recurrent model predicts this downwards movement more correctly.



(a) Seaquest (b) Space Invaders (c) Freeway (d) QBert (e) Ms Pacman

Figure 5: Game play performance using the predictive model as an emulator. 'Emulator' and 'Rand' correspond to the performance of DQN with true frames and random play respectively. The x-axis is the number of steps of prediction before re-initialization. The y-axis is the average game score measured from 30 plays.

Qualitative Analysis of Relative Strengths and Weaknesses of Feedforward and Recurrent Encoding. We hypothesize that feedforward encoding can model more precise spatial transformations because its convolutional filters can learn temporal correlations directly from pixels in the concatenated frames. In contrast, convolutional filters in recurrent encoding can learn only spatial features from the one-frame input, and the temporal context has to be captured by the recurrent layer on top of the high-level CNN features without localized information. On the other hand, recurrent encoding is potentially better for modeling arbitrarily long-term dependencies, whereas feedforward encoding is not suitable for long-term dependencies because it requires more memory and parameters as more frames are concatenated into the input.

As evidence, in Figure 4a we show a case where feedforward encoding is better at predicting the precise movement of the controlled object, while recurrent encoding makes a 1-2 pixel translation error. This small error leads to entirely different predicted frames after a few steps. Since the feedforward and recurrent architectures are identical except for the encoding part, we conjecture that this result is due to the failure of precise spatio-temporal encoding in recurrent encoding. On the other hand, recurrent encoding is better at predicting when the enemies move in Space Invaders (Figure 4b). This is due to the fact that the enemies move after 9 steps, which is hard for feedforward encoding to predict because it takes only the last four frames as input. We observed similar results showing that feedforward encoding cannot handle long-term dependencies in other games.

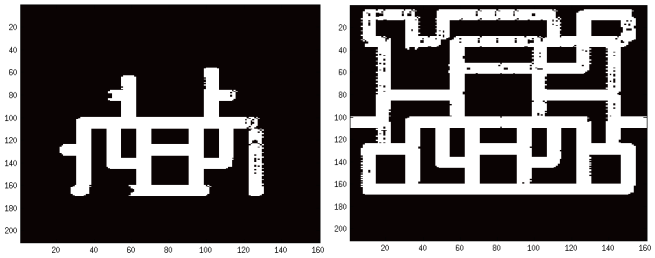
4.2 Evaluating the Usefulness of Predictions for Control

Replacing Real Frames with Predicted Frames as Input to DQN. To evaluate how useful the predictions are for playing the games, we implement an evaluation method that uses the predictive model to replace the game emulator. More specifically, a DQN controller that takes the last four frames is first pre-trained using real frames and then used to play the games based on $\epsilon = 0.05$-greedy policy where the input frames are generated by our predictive model instead of the game emulator. To evaluate how the depth of predictions influence the quality of control, we re-initialize the predictions using the true last frames after every n-steps of prediction for $1 \leq n \leq 100$. Note that the DQN controller never takes a true frame, just the outputs of our predictive models.

The results are shown in Figure 5. Unsurprisingly, replacing real frames with predicted frames reduces the score. However, in all the games using the model to repeatedly predict only a few time

Table 1: Average game score of DQN over 100 plays with standard error. The first row and the second row show the performance of our DQN replication with different exploration strategies.

Model	Seaquest	S. Invaders	Freeway	QBert	Ms Pacman
DQN - Random exploration	13119 (538)	698 (20)	30.9 (0.2)	3876 (106)	2281 (53)
DQN - Informed exploration	13265 (577)	681 (23)	32.2 (0.2)	8238 (498)	2522 (57)



(a) Random exploration. (b) Informed exploration.

Figure 6: Comparison between two exploration methods on Ms Pacman. Each heat map shows the trajectories of the controlled object measured over 2500 steps for the corresponding method.

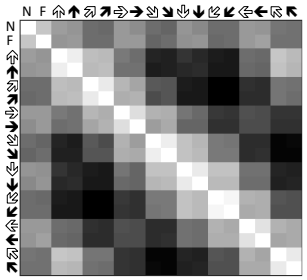


Figure 7: Cosine similarity between every pair of action factors (see text for details).

steps yields a score very close to that of using real frames. Our two architectures produce much better scores than the two baselines for deep predictions than would be suggested based on the much smaller differences in squared error. The likely cause of this is that our models are better able to predict the movement of the controlled object relative to the baselines even though such an ability may not always lead to better squared error. In three out of the five games the score remains much better than the score of random play even when using 100 steps of prediction.

Improving DQN via Informed Exploration. To learn control in an RL domain, exploration of actions and states is necessary because without it the agent can get stuck in a bad sub-optimal policy. In DQN, the CNN-based agent was trained using an ϵ-greedy policy in which the agent chooses either a greedy action or a random action by flipping a coin with probability of ϵ. Such random exploration is a basic strategy that produces sufficient exploration, but can be slower than more informed exploration strategies. Thus, we propose an *informed exploration* strategy that follows the ϵ-greedy policy, but chooses exploratory actions that lead to a frame that has been visited least often (in the last d time steps), rather than random actions. Implementing this strategy requires a predictive model because the next frame for each possible action has to be considered.

The method works as follows. The most recent d frames are stored in a *trajectory memory*, denoted $D = \{\mathbf{x}^{(i)}\}_{i=1}^{d}$. The predictive model is used to get the next frame $\mathbf{x}^{(a)}$ for every action a. We estimate the visit-frequency for every predicted frame by summing the similarity between the predicted frame and the most d recent frames stored in the trajectory memory using a Gaussian kernel as follows:

$$n_D(\mathbf{x}^{(a)}) = \sum_{i=1}^{d} k(\mathbf{x}^{(a)}, \mathbf{x}^{(i)}); \quad k(\mathbf{x}, \mathbf{y}) = \exp(-\sum_{j} \min(\max((x_j - y_j)^2 - \delta, 0), 1)/\sigma) \quad (7)$$

where δ is a threshold, and σ is a kernel bandwidth. The trajectory memory size is 200 for QBert and 20 for the other games, $\delta = 0$ for Freeway and 50 for the others, and $\sigma = 100$ for all games. For computational efficiency, we trained a new feedforward encoding network on 84×84 gray-scaled images as they are used as input for DQN. The details of the network architecture are provided in the supplementary material. Table 1 summarizes the results. The informed exploration improves DQN's performance using our predictive model in three of five games, with the most significant improvement in QBert. Figure 6 shows how the informed exploration strategy improves the initial experience of DQN.

4.3 Analysis of Learned Representations

Similarity among Action Representations. In the factored multiplicative interactions, every action is linearly transformed to f factors ($\mathbf{W}^a\mathbf{a}$ in Equation 4). In Figure 7 we present the cosine similarity between every pair of action-factors after training in Seaquest. 'N' and 'F' corresponds

to 'no-operation' and 'fire'. Arrows correspond to movements with (black) or without (white) 'fire'. There are positive correlations between actions that have the same movement directions (e.g., 'up' and 'up+fire'), and negative correlations between actions that have opposing directions. These results are reasonable and discovered automatically in learning good predictions.

Distinguishing Controlled and Uncontrolled Objects is itself a hard and interesting problem. Bellemare et al. [2] proposed a framework to learn *contingent regions* of an image affected by agent action, suggesting that contingency awareness is useful for model-free agents. We show that our architectures implicitly learn contingent regions as they learn to predict the entire image.

In our architectures, a factor ($f_i = (\mathbf{W}^a_{i,:})^\top \mathbf{a}$) with higher variance measured over all possible actions, $\mathrm{Var}(f_i) = \mathbb{E}_{\mathbf{a}}\left[(f_i - \mathbb{E}_{\mathbf{a}}[f_i])^2\right]$, is more likely to transform an image differently depending on actions, and so we assume such factors are responsible for transforming the parts of the image related to actions. We therefore collected the high variance (referred to as "highvar") factors from the model trained on Seaquest (around 40% of factors), and collected the remaining factors into a low variance ("lowvar") subset. Given an image and an action, we did two controlled forward propagations: giving only highvar factors (by setting the other factors to zeros) and vice versa. The results are visualized as 'Action' and 'Non-Action' in Figure 8. Interestingly, given only highvar-factors (Action), the model predicts sharply the movement of the object controlled by actions, while the other parts are mean pixel values. In contrast, given only lowvar-factors (Non-Action), the model predicts the movement of the other objects and the background (e.g., oxygen), and the controlled object stays at its previous location. This result implies that our model learns to distinguish between controlled objects and uncontrolled objects and transform them using disentangled representations (see [25, 24, 37] for related work on disentangling factors of variation).

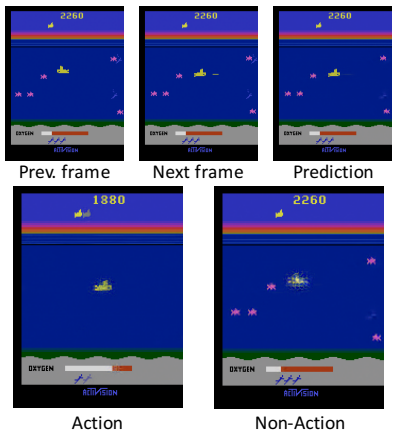


Figure 8: Distinguishing controlled and uncontrolled objects. *Action* image shows a prediction given only learned action-factors with high variance; *Non-Action* image given only low-variance factors.

5 Conclusion

This paper introduced two different novel deep architectures that predict future frames that are dependent on actions and showed qualitatively and quantitatively that they are able to predict visually-realistic and useful-for-control frames over 100-step futures on several Atari game domains. To our knowledge, this is the first paper to show good deep predictions in Atari games. Since our architectures were domain independent we expect that they will generalize to many vision-based RL problems. In future work we will learn models that predict future reward in addition to predicting future frames and evaluate the performance of our architectures in model-based RL.

Acknowledgments. This work was supported by NSF grant IIS-1526059, Bosch Research, and ONR grant N00014-13-1-0762. Any opinions, findings, conclusions, or recommendations expressed here are those of the authors and do not necessarily reflect the views of the sponsors.

References

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[2] M. G. Bellemare, J. Veness, and M. Bowling. Investigating contingency awareness using Atari 2600 games. In *AAAI*, 2012.

[3] M. G. Bellemare, J. Veness, and M. Bowling. Bayesian learning of recursively factored environments. In *ICML*, 2013.

[4] M. G. Bellemare, J. Veness, and E. Talvitie. Skip context tree switching. In *ICML*, 2014.

[5] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[6] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *ICML*, 2009.

[7] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *CVPR*, 2012.

[8] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. In *CVPR*, 2015.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[10] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[11] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang. Deep learning for real-time Atari game play using offline Monte-Carlo tree search planning. In *NIPS*, 2014.

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

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia*, 2014.

[14] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[15] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *ECML*. 2006.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[17] I. Lenz, R. Knepper, and A. Saxena. DeepMPC: Learning deep latent features for model predictive control. In *RSS*, 2015.

[18] R. Memisevic. Learning to relate images. *IEEE TPAMI*, 35(8):1829–1846, 2013.

[19] V. Michalski, R. Memisevic, and K. Konda. Modeling deep temporal dependencies with recurrent grammar cells. In *NIPS*, 2014.

[20] R. Mittelman, B. Kuipers, S. Savarese, and H. Lee. Structured recurrent temporal restricted Boltzmann machines. In *ICML*, 2014.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[23] V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *ICML*, 2010.

[24] S. Reed, K. Sohn, Y. Zhang, and H. Lee. Learning to disentangle factors of variation with manifold interaction. In *ICML*, 2014.

[25] S. Rifai, Y. Bengio, A. Courville, P. Vincent, and M. Mirza. Disentangling factors of variation for facial expression recognition. In *ECCV*. 2012.

[26] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[27] J. Schmidhuber and R. Huber. Learning to generate artificial fovea trajectories for target detection. *International Journal of Neural Systems*, 2:125–134, 1991.

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

[29] I. Sutskever, G. E. Hinton, and G. W. Taylor. The recurrent temporal restricted Boltzmann machine. In *NIPS*, 2009.

[30] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

[31] I. Sutskever, O. Vinyals, and Q. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[32] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[33] G. W. Taylor and G. E. Hinton. Factored conditional restricted Boltzmann machines for modeling motion style. In *ICML*, 2009.

[34] T. Tieleman and G. Hinton. Lecture 6.5 - RMSProp: Divde the gradient by a running average of its recent magnitude. *Coursera*, 2012.

[35] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3D convolutional networks. In *ICCV*, 2015.

[36] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[37] J. Yang, S. Reed, M.-H. Yang, and H. Lee. Weakly-supervised disentangling with recurrent transformations for 3D view synthesis. In *NIPS*, 2015.