Deep Action Conditional Neural Network for Frame Prediction in Atari Games

Elias Wang
Stanford University
elias.wang@stanford.edu

Atli Kosson
Stanford University
akos@stanford.edu

Tong Mu
Stanford University
tongmu@stanford.edu

Abstract

In many problems in computer vision and video frame prediction, such as in robotic control or self driving cars, future frames are not only dependent on past frames, but also on external actions. We study this problem by looking at video frame prediction in the context of Atari 2600 video games where future frames are dependent on past frames as well as actions performed by the players. Based off of previous work such as [12] and [6] we implement a convolutional feedforward model for predicting future frames and rewards as well as a baseline multi-layer perceptron (MLP). We test various training schemes including using an autoencoder to improve prediction clarity and using different loss functions such as the l2 loss, the l1 loss, and the Structural Dissimilarity (DSSIM). We generate data by collecting frames and rewards from a Deep Q Network (DQN) trained to play the game. We found that both quantitatively and qualitatively the convolutional feedforward model achieves better results than the MLP.

1. Introduction

Observing a scene, humans can often predict what will happen over the next couple of seconds. This capability requires some degree of understanding of the content and dynamics of the scene. Video frame prediction is one way to model this behavior and could be an interesting way for unsupervised feature learning. Additionally, video frame prediction can have many other important applications such as in object trajectory prediction and in self driving cars. In many applications of video prediction, future frames are not only dependent on previous frames, but on external inputs or features as well. For example the next observed frame for a robot depends on its input controls.

We study this problem of actioned conditioned video prediction through video frame prediction in the classic Atari 2600 video games. In this setting future frames are dependent on past frames as well as actions performed by the players. We use OpenAI Gym [1] along with a Deep Q Network [10, 11] trained to play the game to collect data and frames from various Atari 2600 video games. We process the data by separating the data into examples which consist of five sequential frames, four as input and the fifth as the target, along with the actions and rewards at each frame. We make predictions of the next frame by inputing this data into a convolutional feedforward model with branches for predicting rewards and the end of the game, a baseline multilayer perceptron, and a model which uses an autoencoder to intialize the weights of the network. We tried the autoencoder as an alternative to the computationally expensive curriculum training done in [12]. During training we experimented with different loss functions in an attempt to achieve better results.

2. Related Work

Recently, there has been much work in video frame prediction. Michalski et al.[9], propose a framework for predicting time series data using recurrent networks. Srivastava et al. [16], use LSTM networks to represent video sequences for various tasks including video frame reconstruction and prediction. Mathieu et al. [8] proposed multiple strategies, including a multi-scale framework, using adversarial training, and a novel loss function using image gradients to achieve better results in frame prediction. Xue et al. [23] propose a probabilistic model for the prediction of future frames, therefore resulting in the ability to sample multiple possible future frames. Walker et al. [18] propose the use of a variational autoencoder to predict the trajectories of pixels in image sequences. Vondrick et al. [17] propose a framework that uses video data with deep learning and recognition algorithms to anticipate actions and objects. Ranzato et al. [14] adapt methods from language modeling to fill missing video frames and predict future frames. Wu et al. [22] propose a generative model, the term ”Galileo”, to predict physical events from video. More recently, Lotter et al.[7] propose a neural network architecture they term ”PredNet” to predict future frames in video sequences. While these works are very powerful for predicting future frames from past frames, they do not account for external features like player actions in Atari games that could affect the future frames.

There has also been work done on video frame prediction that incorporates external actions. Oh et al.[12] propose two deep learning architectures to predict future frames conditioned on actions where actions data is multiplicatively factored in. Additionally Finn et al.[3] propose three deep learning frameworks composed of convolutional LSTMs and convolutional layers for predictions on real world videos conditioned on actions. Leibfried et al. [6] build on the work of Oh et al. by adapting the model to predict future rewards as well.

There has also been much work done on the optimal loss functions for image prediction using convolutional neural networks. Wang et al. [19] propose a new metric to measure the similarity between two images, the structural similarity (SSIM) which attempts to overcome the shortcomings of using the traditional MSE by better mimicking the human visual system. Wang et al. [21] builds on the SSIM metric by proposing a new metric, the multi-scale SSIM (MS-SSIM) that considers the images at multiple scales. Zhao et al. [24] test the effects of various loss functions, including the SSIM and MS-SSIM, in training neural nets and found that a linear combination of the MS-SSIM and the L1 loss was the loss function that gave the best results for their applications. We hypothesize that examining the effect of various loss functions, similar to the procedure of Zhao et al. on the frame and reward prediction of Oh et al. and Leibfried et al. could improve the results.

Finally predicting future rewards with supervised learning has recently been shown to be an effective alternative to traditional temporal difference methods in reinforcement learning in environments with dense rewards. Dosovitskiy and Koltun [2] show that a predictive network can outperform state of art reinforcement learning algorithms on various tasks in the classical first-person shooter game Doom.

3. Methods

3.1. Problem Statement

We consider a setting where we have a game with a good visual representation that can be modeled as Markov Decision Process. We focus on the Atari 2600 games Pong. At each time t we observe a frame $\mathbf{x_t}$, pick an action a_t and then observe the resulting frame $\mathbf{x_{t+1}}$, reward r_{t+1} and a boolean denoting whether the game is over or not d_{t+1}.

We wish to find a function

$$f : \mathbf{x_{1:t}}, a_t \rightarrow \mathbf{x_{t+1}}, r_{t+1}, d_{t+1}$$

that predicts the next frame (at time $t + 1$) from previous frames $\mathbf{x_{1:t}}$. In practice we might only use a few preceding frames for the prediction.

Having accurate estimates of future rewards and when an episode ends could be useful in a number of ways for model based reinforcement learning. It could allow for planning over multiple steps into the future which might also benefit model free agents. By running the agent on the predicted future frames the predictive network might be able to predict mistakes that the agent is about to make and preemptively take control to prevent the mistake from happening. This could be useful for improving the performance of a fully trained agent or used to speed up training by reducing the data necessary.

3.2. Dataset Generation and processing

Our model is trained on a sequence of transitions observed when training a Deep Q Network (DQN) with the structure described in [10] and [11]. We trained a DQN agent[1] on Pong with a epsilon greedy strategy for over 5 million steps. Over the first million steps, the epsilon decays from 1 to 0.1. The training continues for another 4 million frames after that. The DQN makes a decision every 4 frames. We save the first frame and sum the reward over each sequence.

3.3. Model

Core model

Our core model is inspired by [12], and is shown in Figure 1. The feed-forward encoding model consists of four convolutional layers followed by a fully connected layer. The resulting features are then combined multiplicatively with the input actions and fed into the decoding model, which consists of a fully connected layer and four deconvolutional layers that reconstruct the predicted frame. Oh et. al. also suggested a recurrent version of this architecture where the images are fed in one at a time and processed with a LSTM.

The incorporation of multiple input frames to the feed-forward model was done by stacking four sequential image frames along the depth dimension and feeding this as the input to the model. In this way, each filter is able to extract both temporal and spatial information. It is possible to use more or less input frames, but if the dynamics are relatively Markovian, then more frames would not necessarily help. Therefore, we use the standard four frames as commonly used in video prediction literature.

The feed-forward encoding model attempts to extract relevant high-level features from the stacked input frames. This is similar to how later layers (but typically not the last) of AlexNet [5] are used as generic visual features for other tasks. While AlexNet consists of five convolutional layers and two fully connected layers (before the final output layer), our model uses only four convolutional layers and one fully connected layer, each of which is followed by a ReLU nonlinearity. While this reduced capacity may seem problematic, the images from Atari games are much simpler than the ones in ImageNet [15]. Additionally, larger filter

[1] Based on CS234 homework code

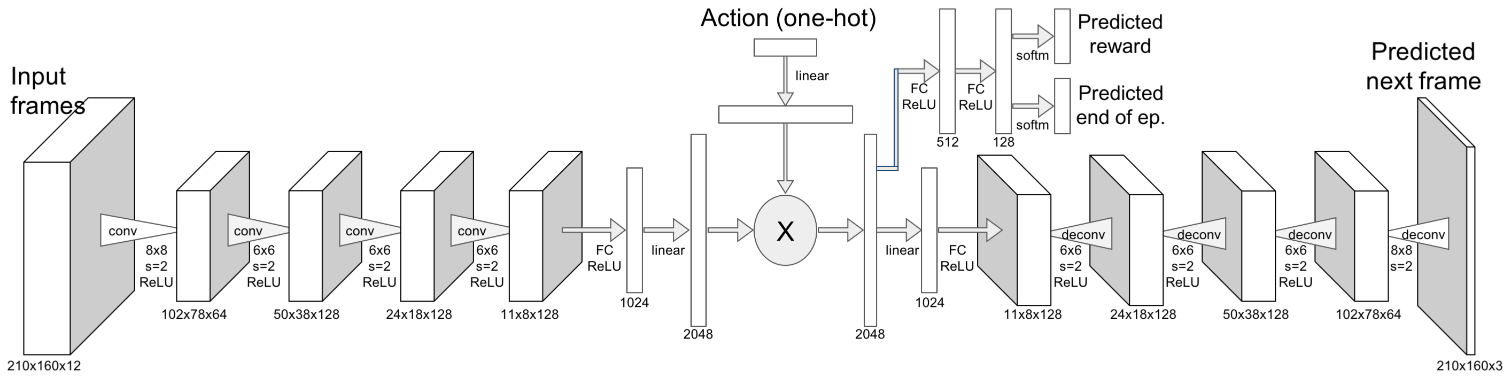


Figure 1: Our core model with reward and end of episode prediction

sizes are used in our model, which may partially alleviate this concern.

Next we combine the encoded features with the action through a multiplicative action-conditional transformation, as proposed by [12]. This can be expressed by Equation 1, where h^{dec} is the output of the action-conditional transformation and the input to the decoding model, h^{enc} is the output of the encoding model, a is a one-hot vector, which essentially acts as an embedding lookup, and the $W's$ are trainable weights.

$$h^{dec} = W^{dec} \left(W^{enc} h^{enc} \odot W^{a} a \right) + b \tag{1}$$

We apply the standard process of deconvolution for generating the predicted image from the decoded features, h^{dec}. The decoding model mirrors the encoding model, and consists of a fully connected layer followed by four deconvolutional layers, with ReLU activations as before. The purpose of the fully connected layer is to reshape the decoded features into the desired ouput shape. The result of the decoding is the final predicted frame.

The model is trained by minimizing the l_2 loss between the predicted frame and the next ground truth frame, as described in Equation 2. Where x_i is the training data and $\hat{x_i}$ is the predicted image for $i = 1, \ldots, N$ for a dataset of size N.

$$\mathcal{L}_i = \frac{1}{2} \|\hat{x_i} - x_i\|^2 \tag{2}$$

In addition we also add a feed-forward network that takes in the features extracted with the encoding network and predicts the reward and done flag associated with the next frame. We will consider basing this network off the architecture described by Dosovitskiy and Koltun [2], which is split into two stems, similar to what is used for the dueling networks by Wang et al [20].

Alternate Loss Functions

The L2 loss has many shortcomings, for example it is very sensitive to outliers and it does not capture how the human visual system perceives images. As a result, we experimented with training our model by minimizing other loss functions as motivated by Zhao et al. [24].

The L1 loss is the sum of the absolute differences between the predicted values and the true values. Compared to the L2 loss, the L1 loss is more robust to outliers as it weights them less.

$$\mathcal{L}_i = |\hat{x_i} - x_i| \tag{3}$$

However we were not able to obtain better results using the L1 loss as compared to the L2 loss. We hypothesize this is caused by the subtleties in training with the L1 loss.

Additionally, we attempted to use the Structural Similarity and Multi-Scale Structural Similarity (SSIM and MS-SSIM) metrics as proposed by Wang et al. [19], [21] as loss functions by using the Structural Dissimilarity (DSSIM). The SSIM metric measures the similarity between two images and better captures how the human visual system perceives images by considering the metrics of local luminescence, structure, and contrast. The MS-SSIM expands on the SSIM and considers the SSIM metric at multiple scales by iteratively applying downsampling and low pass filtering to the image. We use the open source tensorflow implementation SSIM and MS-SSIM provided by Neal Wu [2].

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{4}$$

$$DSSIM(x, y) = \frac{1 - SSIM(x, y)}{2} \tag{5}$$

We were also not able to obtain better results with these metrics. This was due to the fact the SSIM and MS-SSIM

[2] https://github.com/tensorflow/models/blob/master/inception/inception/slim/losses.py

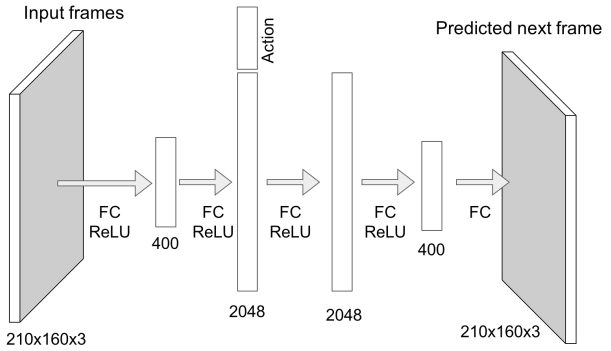


Figure 2: MLP model

metrics themselves have many parameters such as window size and window type used to define locality. Due to the long times needed to train the models, we were not able to tune these parameters to achieve optimal performance within the time constrains.

Reward prediction

We experimented with adding branches to our model to predict rewards and whether or not the next frame corresponds to the end of the episode. We do this by adding a branch from the action conditioned encoding layer that goes through two fully connected layers before two softmax branches corresponding to the reward and episode end. We later discovered that this kind of reward prediction had already been implemented by [6] although they do not predict the end of the episode. Predicting the end of episode could be useful in similar ways as reward prediction.

MLP baseline model

We also test out model against a standard multi-layer perceptron (MLP) baseline, which consists of 4 fully connected layers as suggested by Oh et al. The MLP takes in one frame as input and actions are incorporated by concatenating them onto the output of the second fully connected layer.

4. Datasets and Features

When collecting the data generated by the DQN, we save each frame that has an action. Each frame is stored in full resolution (210 × 160 × 3) along with the corresponding action and reward.

Each example consists of five frames; four frames as input and one frame as the ground truth output. For generating the datasets, we first split all the playthroughs randomly in a 70-15-15 split for the training, validation and testing datasets. Because of the large size of the dataset, consisting of approximately 5 million frames, for each split we randomly sample over all examples and retrieve a portion of the examples. The training set consists of 131,072 examples (of 4 input frames and 1 target frame) which results in 655,360 frames in total. The validation and testing sets both consist of 16,384 examples, or 81,920 frames. Examples of single frames are given in figure 3. Additionally, we calculate the mean image for each split over all the frames in that split and pre-process the image data by subtracting the mean and normalizing by 255 before feeding the data into the models.

5. Experiments and Results

5.1. Core model and variations

Network architectures

The four encoding convolutional layers consisted of 64 (8×8), 128 (6×6), 128 (6×6), 128 (4×4) filters followed by the fully connected layer with 2048 hidden units [12]. The three weight matrices, W^{enc}, W^{a}, and W^{dec}, and the action-conditional transformation all have an output dimension of 2048. The reshaping fully connected decoding layer has hidden dimension 11264 $(11 \times 8 \times 128)$. Finally, the decoding deconvolutional layers have 128 (4×4), 128 (6×6), 128 (6×6), 3 (8×8) filters.

The loss was optimized using the Adam optimizer ($\beta_1 = 0.9, \beta_2 = 0.999$) [4] with a learning rate of 10^{-4} for 10^5 steps with decay. We tested many learning rates to find this optimal one. The convolutional and deconvolutional layers were initialized using Xavier initialization and the fully connected layers for the encoded features and actions were initialized from a random uniform distribution of $[-1, 1]$ and $[-0.1, 0.1]$, respectively. Many of these hyperparameters were adapted from [12], with some adjustments based on our specific dataset and model [3].

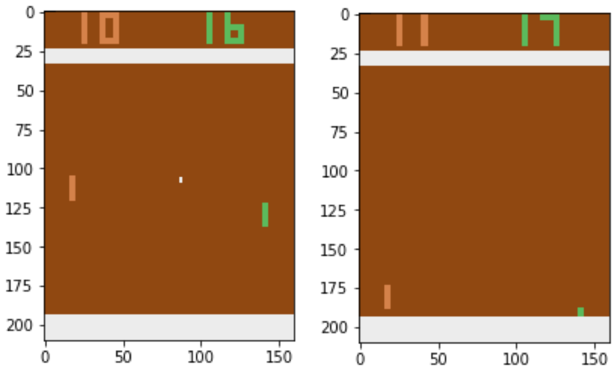


Figure 3: Examples of frames from the game Pong.

[3] Model structure based off Stanford CS224N homework code

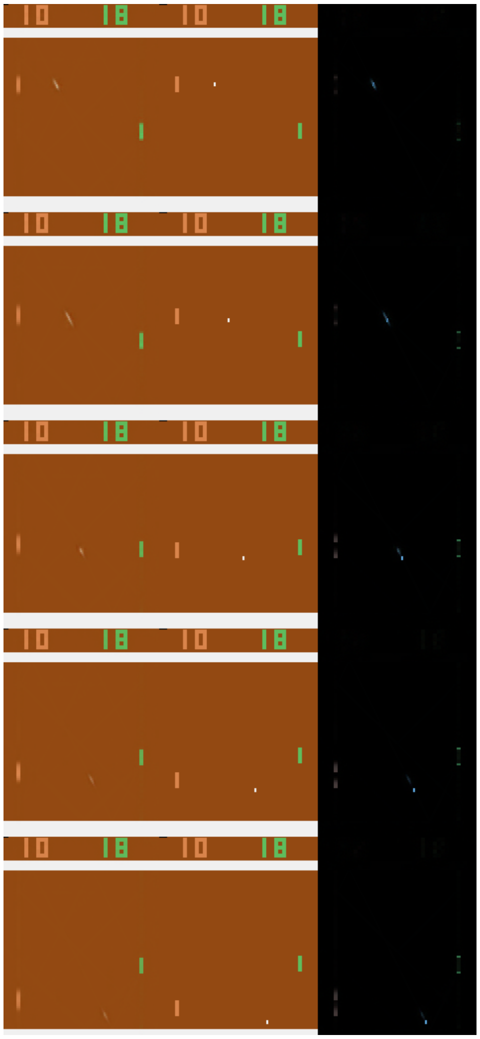
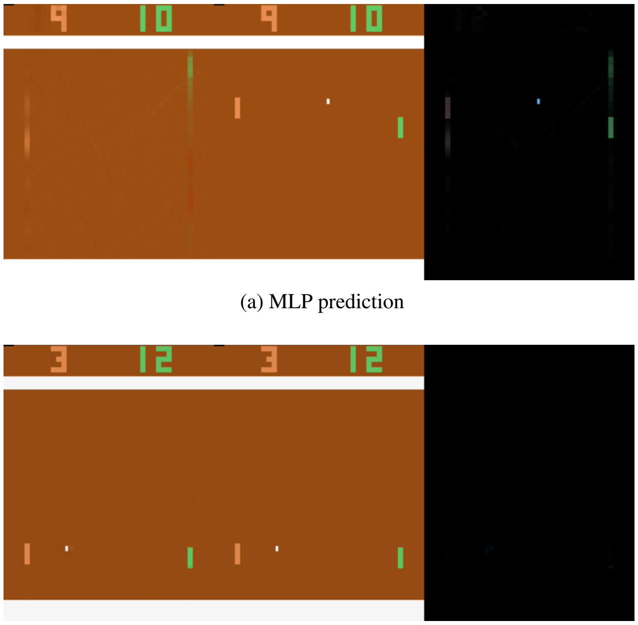


Figure 4: Example of 5-step sequential prediction with core model. Left: Predicted image Middle: Ground truth image Right: Difference

Core model predictions

Although the model was trained on one step prediction, we wanted to see how well it performed on multi-step sequential prediction. This is where we feed in the model's prediction as an input for the next time step. In this case, for the fifth prediction step, none of the input frames are from the actual data, but instead are from the model's prediction. Figure 4 shows the results of this experiment. While the predicted images are not as sharp as the ground truth images, not too surprisingly, we can see that the ball, moving from top-left to bottom-right, and the enemy paddle, moving from top to bottom, relatively track the movements of their counterparts in the ground truth images. The movement of the agent's paddle is less significant, but we can also see that the errors are small there as well.



(a) MLP prediction

(b) Autoencoder transformation

Figure 5: Example of prediction with the MLP model and an encoding decoding transformation for the autoencoder (using the target frame as the input). Left: Predicted image Middle: Ground truth image Right: Difference

Hyperparameter search

We performed a brief hyperparameter search over the learning rate, learning rate decay, and hidden dimension of the fully connected layers. Since the ball is relatively small and occurs in many locations, we hypothesized that increasing the dimension of the feature representations would allow the network to have more capacity to capture the positions of the ball. However, there were no noticeable improvements over values provided in [12].

5.2. MLP baseline model

The baseline MLP has four fully connected layers with dimensions 400, 2048, 2048, and 400. The action was incorporated by concatenating it onto the second fully connected layer.

MLP model predictions

The prediction from the baseline MLP model is qualitatively worse than the feedforward model. As shown in figure 5a, the predicted paddles are very blurry and completely in the wrong location. Additionally, the ball is not predicted at all.

5.3. Reward prediction

In Pong we were able to predict rewards with some limited success. We seemed to be able to predict positive rewards with a much higher F1 score (around 0.8) than negative rewards (0.2). There are likely two issues that cause this. First, sequences with rewards are very rare in the dataset compared to sequences without a reward. Oversampling the reward sequences might fix this but could cause overfitting in the frame prediction because the frames associated with rewards look similar in many ways. Training the reward branch with the rest of the weights fixed could also be a solution. Liebfried et al. [6] encounter similar difficulties but fix it by using a Taylor approximation for the cross entropy loss, but they are looking at different games which could have a very different distribution of rewards. The second issue is an imbalance in the dataset it self. The DQN agent seems to converge to a certain policy in the later stages of training. This results in sequences of frames that occur disproportionally in the dataset which can also lead to overfitting for certain types for frames. The end of episode prediction has similar difficulties, likely because it is a rare event in the training data.

5.4. Autoencoder initialization

We were interested in getting an upper limit for the how well the models could perform so we modified the model to work as an autoencoder, only taking in a single frame and with the same frame as the target. This structure could almost perfectly restore the images, with occasional small errors in the location of the ball.

Since the outputs from the autoencoder (see figure 5b) were much clearer than the outputs of our predictive model we decided to try to use the weights of the autoencoder to create a better predictive model. Our reasoning was that since the autoencoder could already encode and restore the images that we would only need to train a small predictive network in the middle that would take the encoding of the previous frames and calculate the encoding of the next frame which could then be decoded with high quality giving a sharper image than our core model. The model used can be seen in figure 6.

Unfortunately we could not get good performance out of this model. We tried fixing the weights of the encoding and decoding layers and only training the middle layers. This resulted in a loss that would not decrease significantly over the various learning rates that we tested. We also tried letting the model train all the variables, this did not lead to significantly better results than the core model.

This could be because the autoencoder does not learn intuitive features in the image. It seems to have multiple representations for the ball for instance and the same could be true for the paddles. This would probably make the transformation from the encoded input to the target output encoding much more complicated especially because to predict the network could need to access different representation of the objects depending on where they are.

Table 1: Quantitative Comparison of Models. 'Echo' corresponds to just outputting the last input frame as the prediction and serves as a control. 'Autoencoder same' is another control in which an autoencoder with the structure of our model is given the target image and simply has to reproduce it.

Loss	MLP	Feedforward	Echo	Autoencoder same
$\mathcal{L}1$	138.4	74.8	78.6	13.4
$\mathcal{L}2$	11.7	11.1	27.6	0.276

5.5. Quantitative Evaluation of Models

To quantitatively evaluate the performance of the models, we compare the average L1 and L2 difference between the predicted image and the true image for the MLP, feedforward, and Autoencoder models on the test dataset in table 1.

We define the Average L1 and L2 differences between the predicted next image $\hat{x_i}$ and true next image x_i over n test examples as:

$$\mathcal{L}1 = \frac{\sum_{i=1}^{n} |\hat{x_i} - x_i|}{2n} \tag{6}$$

$$\mathcal{L}2 = \frac{\sum_{i=1}^{n} \|\hat{x_i} - x_i\|^2}{2n} \tag{7}$$

From the quantitative results, we can see that the Feedforward model performs significantly better in terms of L1 loss compared to the MLP. The 'Echo' autoencoder gives results with an L1 loss similar to the feedforward network. The Autoencoder same gives much better results as expected as it simply reproduces the input image. L1 loss as a quantitative evaluation metric is appropriate as it is more robust to outliers and weights everything equally.

6. Conclusion and Future Work

We create a dataset of the game Pong, using a DQN agent, for the task of action-conditional video prediction. We found that the feedforward model proposed in [12] was able to perform relatively well on a short multi-step prediction task, on a previously untested game, Pong. While they propose a curriculum training, where the model is trained to predict an increasing number of future frames, we sought alternatives that would be less computationally intensive. In their formulation, they trained for a million steps for each phase, with three phases in total.

We propose to initialize the network with the weights learned by training an autoencoder. While the autoencoder

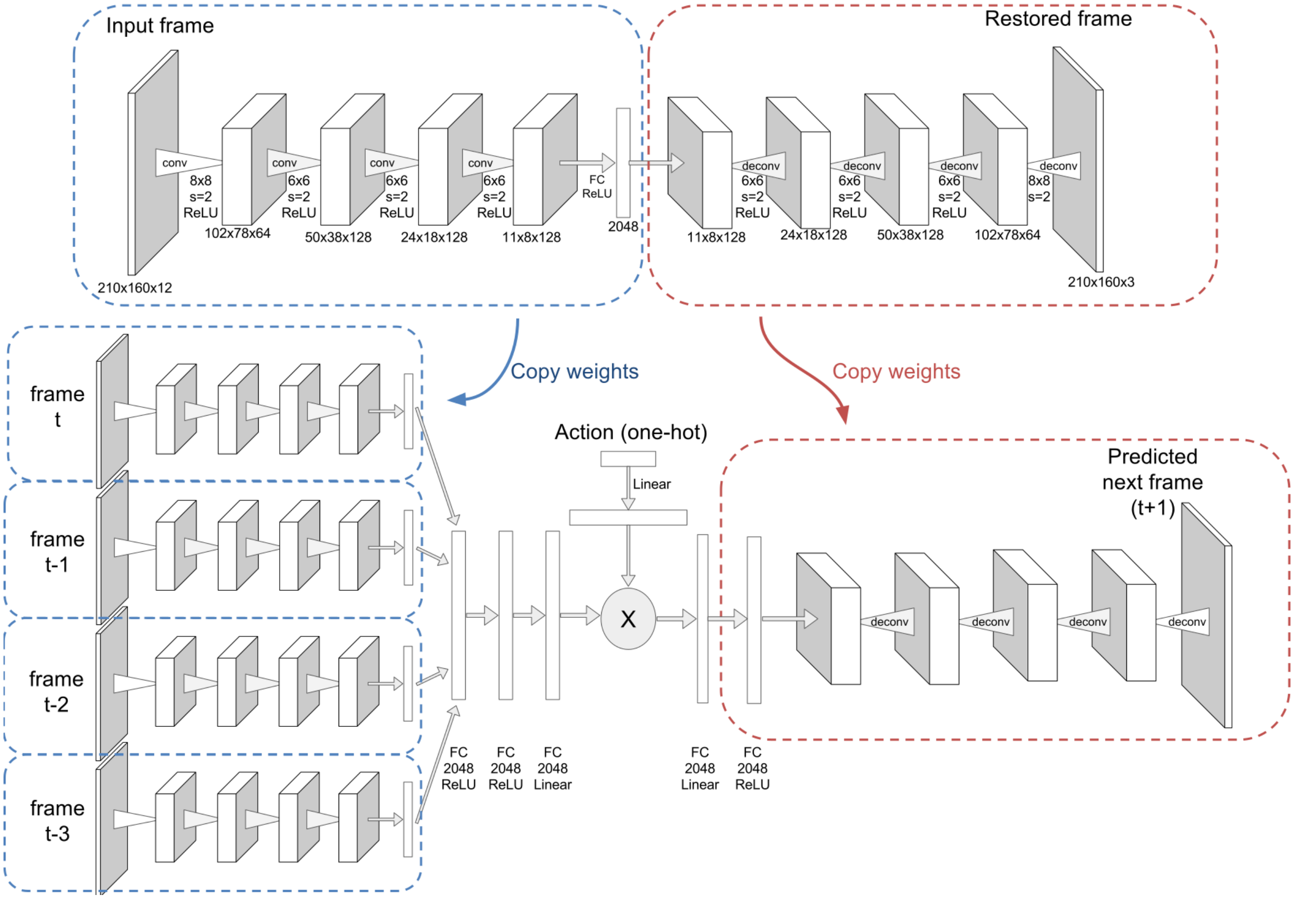


Figure 6: Autoencoder initialization. The network on top is an autoencoder that is trained on with the target frame as an output. After the autoencoder is fully trained the encoding and decoding layers are used to initialize a predictive model.

was able to train very well, the various methods of using the learned weights to bootstrap the video prediction training did not show significant improvements. This may support the conclusion in [12], that the curriculum approach is necessary for stable training. However, it could also be possible that better results could be obtained with a better dataset.

The DQN agent seems to be prone to converging to a single rigid policy when trained on Pong, perhaps due to the limited stochasticity of Pong. This causes a number of imbalance issues with the dataset. This could likely be addressed to some extent by training with more stochasticity, a higher final ϵ value or using different exploration strategy such as the randomized value functions proposed by Osband et al. [13].

In the future we would be interested in running the predictive network and the DQN agent in parallel. Even after 5 million steps of training on Pong the DQN agent will still make mistakes. If the predictive network can accurately predict future frames and rewards we could simulate the next couple of steps of the DQN agent at every time point. If we predict a mistake, which coorespond to negative rewards in Pong, we could then try to search for a sequence of actions (again through simulation with our predictive network) that will prevent the mistake. After the mistake has been averted the DQN agent would resume control. If the training time for the predictive network could be reduced it would also be interesting to train both networks in parallel and see whether this sort of intervention could speed up the learning of the DQN agent.

References

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym, 2016.

[2] A. Dosovitskiy and V. Koltun. Learning to act by predicting the future. *arXiv preprint arXiv:1611.01779*, 2016.

[3] C. Finn, I. Goodfellow, and S. Levine. Unsupervised learning for physical interaction through video prediction. In *Advances in Neural Information Processing Systems*, pages 64–72, 2016.

[4] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[6] F. Leibfried, N. Kushman, and K. Hofmann. A deep learning approach for joint video frame and reward prediction in atari games. *arXiv preprint arXiv:1611.07078*, 2016.

[7] W. Lotter, G. Kreiman, and D. Cox. Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104*, 2016.

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

[9] V. Michalski, R. Memisevic, and K. Konda. Modeling deep temporal dependencies with recurrent grammar cells"". In *Advances in neural information processing systems*, pages 1925–1933, 2014.

[10] 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.

[11] 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.

[12] J. Oh, X. Guo, H. Lee, R. L. Lewis, and S. Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pages 2863–2871, 2015.

[13] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. In *Advances In Neural Information Processing Systems*, pages 4026–4034, 2016.

[14] M. Ranzato, 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.

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[16] N. Srivastava, E. Mansimov, and R. Salakhudinov. Unsupervised learning of video representations using lstms. In *International Conference on Machine Learning*, pages 843–852, 2015.

[17] C. Vondrick, H. Pirsiavash, and A. Torralba. Anticipating the future by watching unlabeled video. *arXiv preprint arXiv:1504.08023*, 2015.

[18] J. Walker, C. Doersch, A. Gupta, and M. Hebert. An uncertain future: Forecasting from static images using variational autoencoders. In *European Conference on Computer Vision*, pages 835–851. Springer, 2016.

[19] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[20] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[21] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, volume 2, pages 1398–1402. IEEE, 2003.

[22] J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *Advances in neural information processing systems*, pages 127–135, 2015.

[23] T. Xue, J. Wu, K. Bouman, and B. Freeman. Visual dynamics: Probabilistic future frame synthesis via cross convolutional networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2016.

[24] H. Zhao, O. Gallo, I. Frosio, and J. Kautz. Loss functions for neural networks for image processing. *arXiv preprint arXiv:1511.08861*, 2015.