Video Prediction with Action Feedback

Pascal Pompey, Sudeep Sudhir Jain, Andrei Bajenov

CS231n, Stanford

Topic

- Visual prediction of future events based on past observations using CNN and RNN models.
- Many uses, including self-driving cars, robotics, etc.

Problem Setting

- Solving the general problem is very hard. There are many variables at play, and making such a model is unfeasible given our resources.
- We simplify the problem by solving it for a much simpler environment: the Atari Pac-Man game.
- Pac-Man is a good candidate because it is: (1) Relatively complex and (2) Has some elements of uncertainty (e.g. ghosts).
- It is also a fairly deterministic game in that one can predict the next frame given the current frame and the game state.

Data Generation

- We use the OpenAI Gym environment to generate a time series of in-game images. These images are then used for training and evaluation.
- The Pac-Man is fed random actions to generate the gameplay.
- We attempt to predict the next frame of the image given the previous sequence of frames and the current action taken.

Baseline

Use the current frame as the prediction for the next frame.

- In Pac-Man, changes between consecutive frames are very minimal, making this a very hard baseline to beat. The prediction of the game environment is perfectly sharp, so the only observed error is in the position of the ghosts and the Pac-Man.
- RMSE tends to generate blurry images, which makes it even harder to design models that would beat the sharp baseline.

Architectural Considerations

First observation:

- Predicting the difference is easier than predicting the whole frame. The final model's output is subtracted from the original image:
  `modelOutput = neuralNet(output) - img`
- This method enabled us to generate images close to the baseline. It is easier for the model to predict a blank image than a complex one.

Second observation:

- The core objects in the game have a given size and evolved in a world of paths delimited by walls. This means that, to be able to understand what is happening, the final layers of a CNN need to have a minimum receptive field of at least the object and its surrounding walls.
- After some tests, it was concluded that the minimal receptive field to see Pac-Man or a ghost along with its two closest walls was 13 pixels. 13 pixels is therefore the lower bound for the receptive field of our final encoding CNN layers.

Model 1: Purely Forward CNN

Model Description:

- Convolutional layers with filter size 3 and 120 filters conserving the original image size (padding 1, stride 1) are stacked together to reach the intended receptive field of 13.
- Then ConvTranspose layers with filter size 3 conserving the original image size (padding 1, stride 1) are stacked together until the final layer generates an image.

Results:

This model was able to get results that were slightly (yet not significantly) better than the baseline. Observing the generated images and their difference with the target revealed that:

- The model was able to capture the shapes of the Pac-Man and ghosts
- The RMSE criterion was pushing the model to mean values in areas of high uncertainty, resulting to more blurry images around the Pac-Man, ghosts or blinking objects.

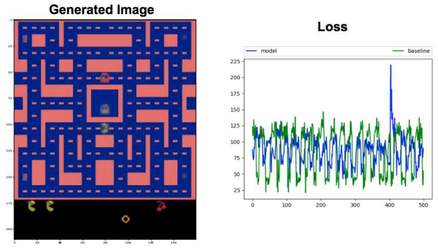

Figure 1: Model 1 Generated Image and Loss

Model 2: Reducing Auto-Encoder

Model Description:

A CNN encoder divides the size of the image by 2 in the height and width, until a receptive field of at least 13 is reached. To do this, two methods are used:

1. Stride of 2
2. Max Pooling Layers

A ConvTranspose decoder scales up the generated fields until the size of the original image is recreated.

Results:

The reducing auto-encoder achieved slightly worse accuracy than the purely forward CNN and the baseline, but:

- It did generate much sharper images than the purely forward CNN.
- It did demonstrate that it was possible to reduce the number of parameters substantially while still being able to regenerate good images.

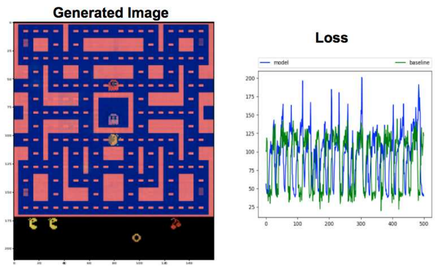

Figure 2: Model 2 Generated image and Loss

Model 3: Reducing Encoder alongside an RNN

Model Description:

Pac-Man follows the Markov property. Based on a suitable state and the current image, it is possible to fully determine the next image and state. It is therefore perfectly legitimate to try and use some RNNs as core components of the architecture.

RNNs contain fully connected layers and are therefore very memory intensive. This means that using them without triggering out of memory errors requires aggressively scaling down the original image of (3*210*160) to a much smaller parameters set.

The architecture was therefore to:

1. Use a convolutional encoder to reduce the original image to a reasonable size.
2. Apply the RNN on the flattened output of that image.
3. Use a convolutional decoder to reconstruct the image from the RNNs updated state

Results:

We were not able to achieve a loss close to the baseline. With a bit more parameter tuning we are hoping that this model will be able to beat the baseline.

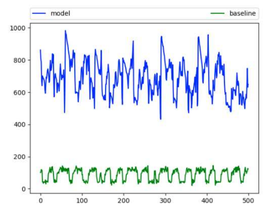

Figure 3: Model 3 Loss

References

[1] Mathieu, Michael, Camille Couprie, and Yann LeCun. "Deep multi-scale video prediction beyond mean square error." arXiv preprint arXiv:1511.05440 (2015).

[2] Oh, Junhyuk, et al. "Action-conditional video prediction using deep networks in atari games." Advances in Neural Information Processing Systems . 2015.

[3] Bengio, Yoshua, et al. "Generalized denoising auto-encoders as generative models." Advances in Neural Information Processing Systems . 2013.