# Frame Skipping and Pre-Processing for Deep Q-Networks on Atari 2600 Games

Nov 25, 2016

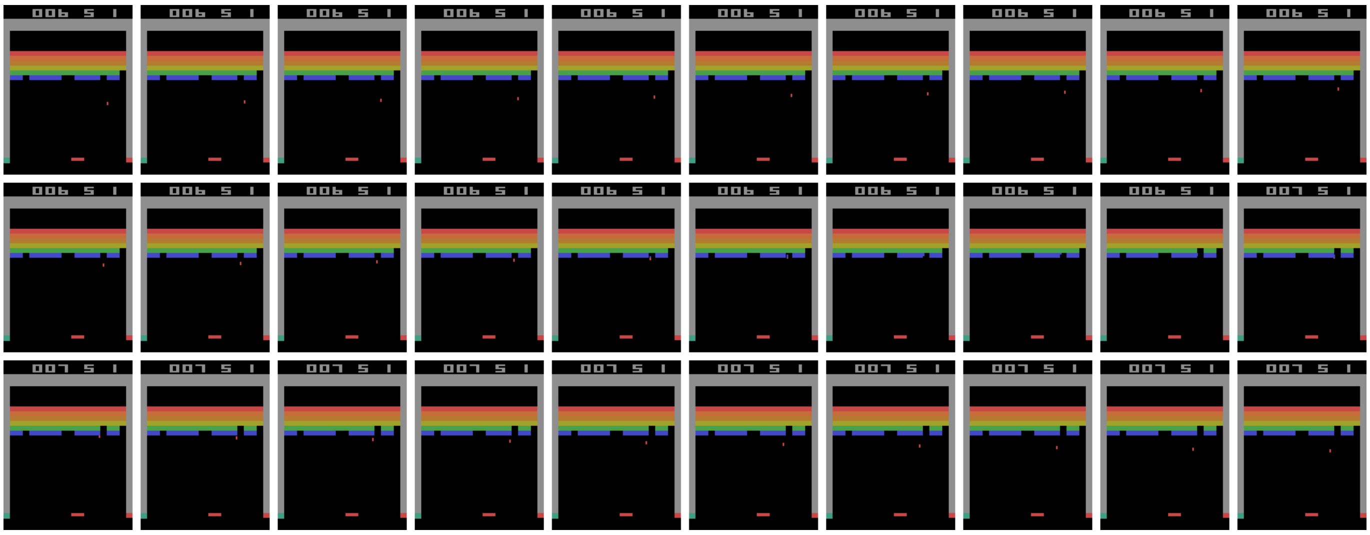
For at least a year, I've been a huge fan of the Deep Q-Network algorithm. It's from Google DeepMind, and they used it to train AI agents to play classic Atari 2600 games at the level of a human *while only looking at the game pixels and the reward*. In other words, the AI was learning just as we would do!

Last year, I started a personal project related to DQNs and Atari 2600 games, which got me delving into the details of DeepMind's two papers (from NIPS workshop 2013 and NATURE 2015). One thing that kept confusing me was how to interpret their frame skipping and frame processing steps, because each time I read their explanation in their papers, I realized that their descriptions were rather ambiguous. Therefore, in this post, I hope to clear up that confusion once and for all.

I will use Breakout as the example Atari 2600 game, and the reference for the frame processing will be from the NATURE paper. Note that the NATURE paper is actually rather "old" by deep learning research standards (and the NIPS paper is ancient!!), since it's missing a lot of improvements such as Prioritized Experience Replay and Double Q-Learning, but in my opinion, it's still a great reference for learning DQN, particularly because there's a great open-source library which implements this algorithm (more on that later).

To play the Atari 2600 games, we generally make use of the Arcade Learning Environment library which simulates the games and provides interfaces for selecting actions to execute. Fortunately, the library allows us to extract the game screen at each time step. I modified some existing code from the Python ALE interface so that I could play the Atari games myself by using the arrow keys and spacebar on my laptop. I took 30 consecutive screenshots from the middle of one of my Breakout games and stitched them together to form the image below. It's a bit large; right click on it and select "Open

Image in New Tab" to see it bigger. The screenshots should be read left-to-right, up-to-down, just like if you were reading a book.



Note that there's no reason why it had to be *me* playing. I could have easily extracted 30 consecutive frames from the AI playing the game. The point is that I just want to show what a sequence of frames would look like.

I saved the screenshots each time I executed an action using the ALE Python interface. Specifically, I started with `ale = ALEInterface()` and then after each call to `ale.act(action)`, I used `rgb_image = ale.getScreenRGB()`. Thus, the above image of 30 frames corresponds to *all* of the screenshots that I "saw" while playing in the span of a half-second (though obviously no human would be able to distinguish among all 30 frames at once). In other words, if you save each screenshot after every action gets executed from the ALE python library, there's *no* frame skipping.

Pro tip: if you haven't modified the default game speed, play the game yourself, save the current game screenshot after every action, and check if the total number of frames is roughly equivalent to the number of seconds multiplied by 60.

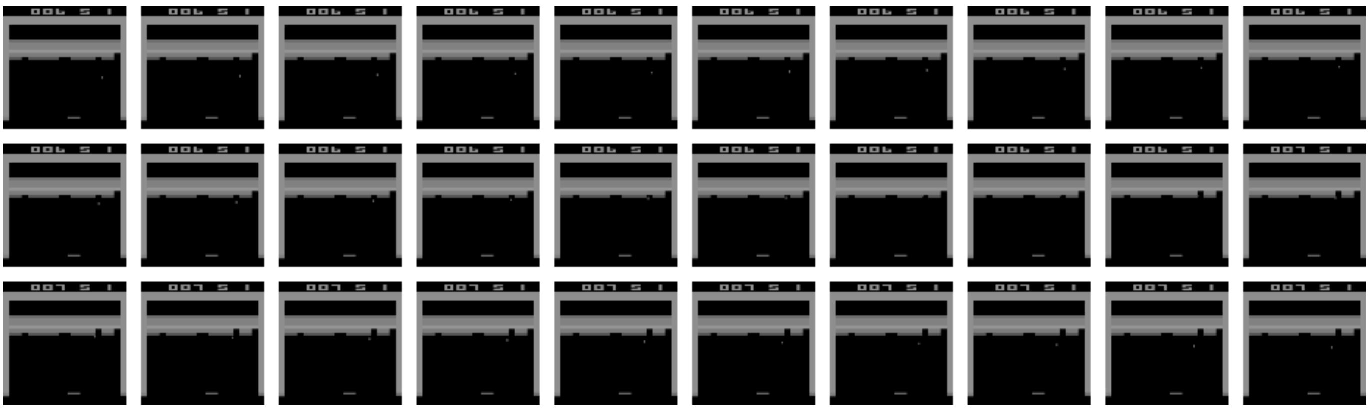
Now that we have 30 consecutive in-game images, we need to process them so that they are not too complicated or high dimensional for DQN. There are two basic steps to this process: *shrinking* the image, and converting it into *grayscale*. Both of these are not as straightforward as they might seem! For one, how do we shrink the image? What size is a good tradeoff for computation time versus richness? And in the particular case of Atari 2600 games, such as in Breakout, do we want to crop off the score at the top, or leave it in? Converting from RGB to grayscale is also – as far as I can tell – an undefined problem, and there are different formulas to do the conversion.

For this, I'm fortunate that there's a *fantastic* DQN library open-source on GitHub called deep_q_rl, written by Professor Nathan Sprague. Seriously, it's awesome! Professor Sprague must have literally gone through almost every detail of the Google DeepMind source code (also open-source, but harder to read) to replicate their results. In addition, the code is extremely flexible; it has separate NIPS and NATURE scripts with the appropriate hyperparameter settings, and it's extremely easy to tweak the settings.

I browsed the deep_q_rl source code to learn about how Professor Sprague did the downsampling. It turns out that the NATURE paper did a linear scale, thus *keeping* the scores inside the screenshots. That strikes me as a bit odd; I would have thought that cropping the score entirely would be better, and indeed, that seems to have been what the NIPS 2013 paper did. But whatever. Using the notation of (height,width), the final dimensions of the downsampled images were (84,84), compared to (210,160) for the originals.

To convert RGB images to grayscale, the deep_q_rl code uses the built-in ALE grayscale conversion method, which I'm guessing DeepMind also used.

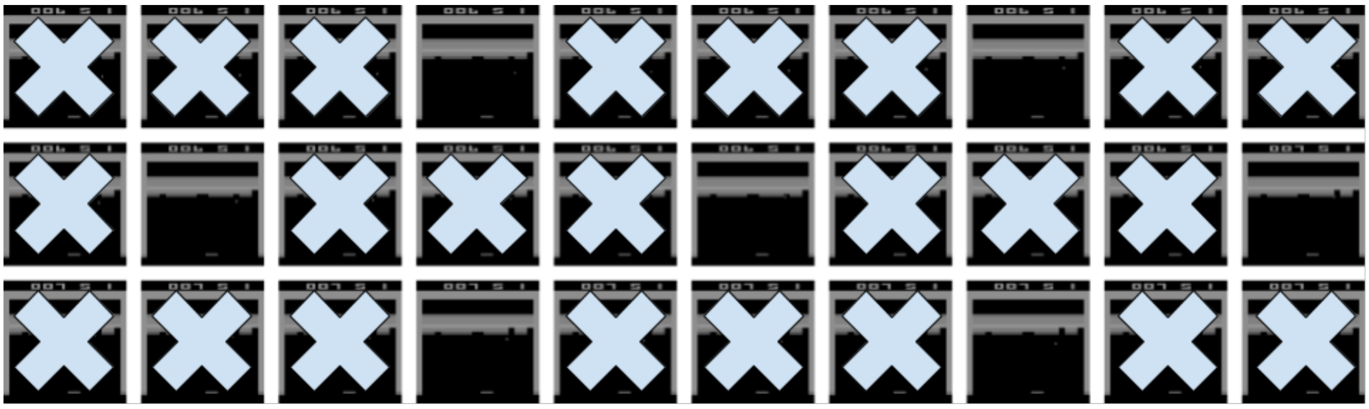
The result of applying these two pre-processing steps to the 30 images above results in the new set of 30 images:



Applying this to *all* screenshots encountered in gameplay during DQN training means that the data dimensionality is substantially reduced, and also means the algorithm doesn't have to run as long. (Believe me, running DQN takes a *long* time!)

We have all these frames, but what gets fed as *input* to the "Q-Network"? The NATURE and NIPS paper used a sequence of *four* game frames stacked together, making the data dimension (4,84,84). (This parameter is called the "phi length") The idea is that the action agents choose depends on the prior sequence of game frames. Imagine playing Breakout, for instance. Is the ball moving up or down? If the ball is moving down, you better get the paddle in position to bounce it back up. If the ball is moving up, you can wait a little longer or try to move in the opposite direction as needed if you think the ball will eventually reach there.

This raises some ambiguity, though. Is it that we take every four consecutive frames? NOPE. It turns out that there's a *frame skipping* parameter, which confusingly enough, the DeepMind folks *also* set at 4. What this means in practice is that only every fourth screenshot is considered, and *then* we form the "phi"s, which are the "consecutive" frames that together become the input to the neural network function approximator. Consider the updated sequence of screenshots, with the skipped frames denoted with an "X" over them:



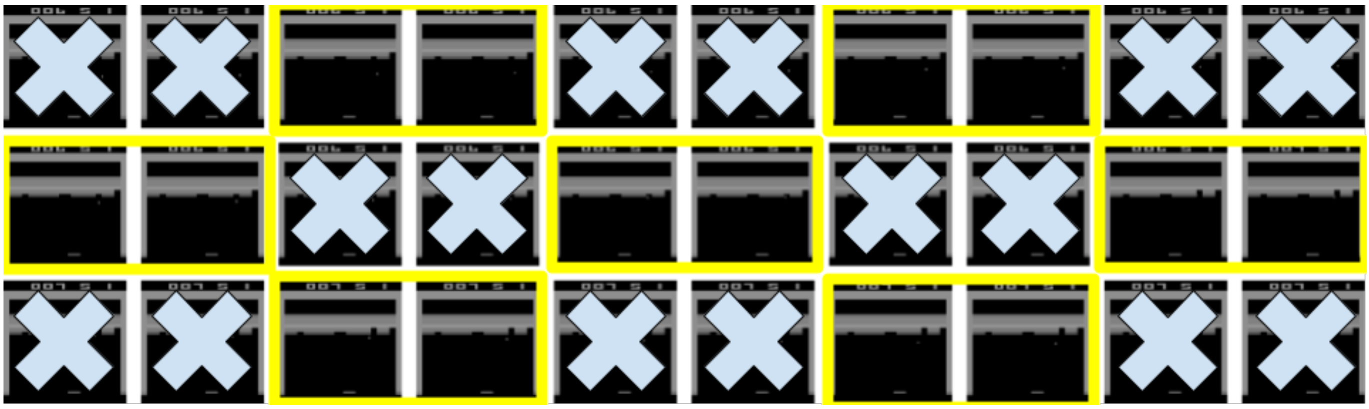
What ends up happening is that the states are four consecutive frames, *ignoring* all of the "X"-ed out screenshots. In the above image, there are only seven non-skipped frames. Let's denote these as $x_1, x_2, \dots, x_7$. The DQN algorithm will use $s_1 = (x_1, x_2, x_3, x_4)$ as one state. Then for the next state, it uses $s_2 = (x_2, x_3, x_4, x_5)$. And so on.

Crucially, *the states are overlapping.* This was another thing that wasn't apparent to me until I browsed Professor Sprague's code. Did I mention I'm a huge fan of his code?

A remark before we move on: one might be worried that the agent is throwing away a lot of information using the above procedure. In fact, it actually makes a lot of sense to do this. Seeing four consecutive frames without subsampling doesn't give enough information to discern motion that well, especially after the downsampling process.

There's one more obscure step that Google DeepMind did: they took the component-wise maximum over two consecutive frames, which helps DQN deal with the problem of how certain Atari games only render their sprites every other game frame. Is this maximum done over *all* the game screenshots, or only the *subsampled* ones (every fourth)? It's the former.

Ultimately, we end up with the revised image below:



I've wrapped two consecutive screenshots in yellow boxes, to indicate that we're taking the pixel-by-pixel (i.e. component-wise) maximum of the two images, which then get fed into as components for our states $s_i$. Think of each of the seven yellow boxes above as forming *one* of the $x_i$ frames. Then everything proceeds as usual.

Whew! That's all I have to say regarding the pre-processing. If you want to be consistent with the NATURE paper, try to perform the above steps. There's still a lot of ambiguity, unfortunately, but hopefully this blog post clarifies the major steps.

Aside from what I've discussed here, I want to bring up a few additional points of interest:

- OpenAI also uses the Atari 2600 games as their benchmark. However, when we perform an action using OpenAI, the action we choose is performed either 2, 3, or 4 frames in a row, *chosen at random*. These are at the granularity of a single, true game frame. To clarify, using our method above from the NATURE paper would mean that each action that gets chosen is repeated four times (due to four skipped game frames). OpenAI instead uses 2, 3, or 4 game frames to introduce stochasticity, but *doesn't* use the maximum operator across two consecutive images. This means if I were to make a fifth image representing the frames that we keep or skip with OpenAI, it would look like the third image in this blog post, except the consecutive X's would number 1, 2, or 3 in length, and not just 3. You can find more details in this GitHub issue.
- On a related note regarding stochasticity, ALE has an *action repeat probability* which is hidden to the programmer's interface. Each time you call an action, the ALE engine will *ignore* (!!) the action with probability 25% and simply repeat the previous action. Again, you can find more discussion on the related GitHub issue.
- Finally, as *another* way to reduce stochasticity, it has become standard to use a *human starts* metric. This method, introduced in the 2015 paper *Massively Parallel Methods for Deep Reinforcement Learning*, suggests that humans play out the initial trajectory of the game, and then the AI takes over from there. The NATURE paper did

something similar to this metric, except that they chose a random number of no-op actions for the computer to execute at the start of each game. That corresponds to their "no-op max" hyperparameter, which is described deep in the paper's appendix.

Why do I spend so much time obsessing over these details? It's about understanding the problem better. Specifically, what *engineering* has to be done to make reinforcement learning work? In many cases, the theory of an algorithm breaks down in practice, and substantial tricks are required to get a favorable result. Many now-popular algorithms for reinforcement learning are based on similar algorithms invented decades ago, but it's only now that we've developed the not just the computational power to run them at scale, but also the engineering tricks needed to get them working.

**8 Comments** seitasplace 1 Login

Recommend 2 Share Sort by Oldest

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

**Kerawit Somchaipeng** • 7 months ago

I enjoyed reading your post so much becase I'm asking myself the same questions right now as I'm trying to implement my own version of dqn. Thanks to you, I found many of the answers here.

There is also another small detail that both original papers didn't discuss at all which is how they manage rewards returned from the emulator during skipped frames. Do you have the implementation detail on that as well?

• Reply • Share ›

**Daniel Seita** Mod Kerawit Somchaipeng • 7 months ago

I'm glad you liked this.

Off the top of my head, I don't know about details on rewards from skipped frames. My guess is those are ignored but don't quote me on this.

• Reply • Share ›

**yobibyte** ➦ Daniel Seita • 3 months ago

They return the cumulative reward.

• Reply • Share ›

**Daniel Seita** Mod ➦ yobibyte • 3 months ago

This makes sense to me.

• Reply • Share ›

**yobibyte** ➦ Daniel Seita • 3 months ago

Agree.

• Reply • Share ›

**Yiding Yu** • 3 months ago

Dear Seita,
Thanks for interpreting of the preprocessing phase of nature DQN paper, it really helps to understand the hided details in the paper. I agree the state has a fixed length with 4 screen x_t, but it is also related to the actions you take between the 4 screens, e.g., s_t = x_t-3, a_t-3, x_t-2, a_t-2, x_t-1, a_t-1, x_t, this is a little different from your blog s1=(x1,x2,x3,x4). What's your idea?

• Reply • Share ›

**Daniel Seita** Mod ➦ Yiding Yu • 3 months ago

As far as I remember, we're not combining states and actions together. The input to the Q-network should be (x1,x2,x3,x4). The output should be a softmax indicating the action we should pick right after seeing x4.

• Reply • Share ›

**Sajad Norouzi** • 15 days ago

Hey, thanks for your great article. it seems gym added some new environments and problem of skipped frames has been sovled.
look at NoFrameskip-v4:
https://github.com/openai/g...

• Reply • Share ›

ALSO ON **SEITASPLACE**

**The Four Classes That I Have Self-Studied**

2 comments • a year ago•

Daniel Seita — Yong,Yes, I looked at Andrew Ng's machine learning course, but

**Going Deeper Into Reinforcement Learning: Understanding Q-Learning ...**

2 comments • 10 months ago•

Daniel Seita — Good to know that the course will be offered again! I should

Andrew Ng's machine learning course, but one needs to make the distinction …

**IPython, Jupyter Notebooks, and matplotlib**

2 comments • 2 years ago•

**Ricardo de Azambuja** — Hi, just to add a new information: instead of "%matplotlib inline", if you have Matplotlib 1.5 or …

course will be offered again. I should update my review of CS 294-112 and …

**Thoughts on Isolation: Why I Hated the Fall 2013 and Fall 2015 Semesters**

1 comment • 2 years ago•

**Yongyi Chen** — It's so great to read this and know I'm not alone in having this problem. (And you're not alone, either!) …

Subscribe  Add Disqus to your siteAdd DisqusAdd  Privacy

## Seita's Place

Seita's Place
seita@berkeley.edu

DanielTakeshi
(Never!)

This is my blog, where I have written over 250 articles on a variety of topics, most of which are about one of two major themes. The first is computer science, which is my area of specialty as a Ph.D. student at UC Berkeley. The second can be broadly categorized as "deafness," which relates to my experience and knowledge of being deaf.