

GuessTuples Project

Andrew J. Wren

28 April 2021

Abstract

Notes on GuessTuples project

1 Configuring the nets

1.1 Alice

One per bit. The input array to guess is $\mathbf{x} = (x_j)_{j=0,\dots,N_{\text{elements}}}$. There should be N_{code} outputs taking values $\mathbf{y} = (y_j)_{j=0,\dots,N_{\text{code}}}$.

Normalise all the rewards so that for each bit j , $r_{j\checkmark} + (N_{\text{code}} - 1) r_{j\text{x}} = 0$. In other words

$$r_{jk} \leftarrow r_{jk} - \frac{r_{j\checkmark} + (N_{\text{code}} - 1) r_{j\text{x}}}{N_{\text{code}}}. \quad (1)$$

The Q estimate is then taken to be

$$Q(\mathbf{x}) = \sum_j b_j y_j \equiv \sum_j |y_j|, \quad (2)$$

where

$$b_j = \text{sgn}(y_j) \quad (3)$$

is the prediction for the machine value of the j th bit. The loss function is

$$L = |Q(\mathbf{x}) - r|^2. \quad (4)$$

Alternative approaches include:

1. Two outputs for each bit showing the reward for each of 0 and 1. *May reflect negative rewards better?*
2. Combine the rewards from the bits (with either one or two outputs per bit) by something other than addition - e.g. multiplication or via an NN. *The NN option seems quite interesting. Interesting to use pytorch's gradients for that.*
3. **One per code.** One output for each possible code. *Might work but $2^{N_{\text{code}}}$ is quite large... not impossibly so if $N_{\text{code}} = 8$.*

4. Inspired by Ref. [1], feed \mathbf{x} into Alice's 'first' net, to get output \mathbf{y} , and all possible codes \mathbf{c} into her 'second' net, both net's having the same target dimensionality (a hyperparameter). Then the code to use is the one $\mathbf{c}(\mathbf{x})$ closest to the output of the first net, with the Q being given by the inner product $Q = \langle \mathbf{y}, \mathbf{c}(\mathbf{x}) \rangle$. Ref. [2] might provide an alternative, actor-critic, approach on a similar theme. The main case above is, in effect, an embedding of \mathbf{x} into the target space (of dimensionality N_{code}) which then compares with the natural embedding of \mathbf{c} by, in effect, the inner product.
5. Ref. [3] suggest sequentialising, which points to a variant of our main approach which does each bit in succession and feeding those results into successive Alice-nets so the Q -estimate for later bits takes account of earlier bits / estimates, with the N_{code} th estimate providing a final code \mathbf{c} and Q -estimate for that code.
6. Move away from typical Q-learning. Instead Alice's output is the code \mathbf{c} and then when Bob makes his choice \mathbf{x}_{pred} (see below) run that choice through a copy of Alice, to get \mathbf{c}_{Bob} and then the loss function is

$$L = -r(\mathbf{c}, \mathbf{c}_{\text{Bob}}). \quad (5)$$

1.2 Bob

One per bit aka Simple. Bob receives a matrix, $\mathbf{X} = (X_i) = (X_{ij})$ for $0 \leq i < N_{\text{select}}$, $0 \leq j < N_{\text{elements}}$, and a code $\mathbf{c} = (c_k)_{k=0, \dots, N_{\text{code}}}$. Why not makes his outputs be Q -estimates $\mathbf{z} = (z_i)_{i=0, \dots, N_{\text{select}}}$. Bob's prediction is then $\mathbf{x}_{\text{pred}} = \mathbf{X}_{i_{\text{pred}}}$ where

$$i_{\text{pred}} = \text{argmax}_i (z_i). \quad (6)$$

The loss function is

$$L = |z_{i_{\text{pred}}} - r|^2. \quad (7)$$

How do we enforce covariance with respect to the order of (X_i) ?

1. Covariance will occur naturally and quickly without any specific intervention. *To be determined.*
2. Covariance can be enforced through choosing a set $\{\sigma\} \subseteq S_{n_{\text{code}}}$, which could be generated element-by-element by composing randomly-selected basis transpositions $(j \ j+1)$, and then adding to the loss a term

$$\mu \sum_{\sigma} |z - \sigma^{-1} [z(\sigma[\mathbf{X}])]|^2 \quad (8)$$

for some fixed hyperparameter $\mu > 0$. Note this the term is still run backward through the original $\mathbf{x} \mapsto \mathbf{z}$ net configuration only. *How effective would that be? How big does $\{\sigma\}$ have to be? And how much time would the permutation and the additions forward passes cost?*

3. Enforce covariance via direct identification of weights in Bob's net. *How?*
4. Something related to set transformers. ?
5. Adopt a different basic set-up where each (X_i) is fed through the net separately, alongside the code \mathbf{c} , resulting in a Q -estimate z_i . Then find the loss function as in Eq. 7. *Seems the most straightforward?*

None of these quite amount to Bob seeks to reproduce the Alice's code vocabulary. However Bob could additionally set up a net in the same basic configuration as Alice's (he doesn't know the weights of course) and train *that* net jointly with his main net.

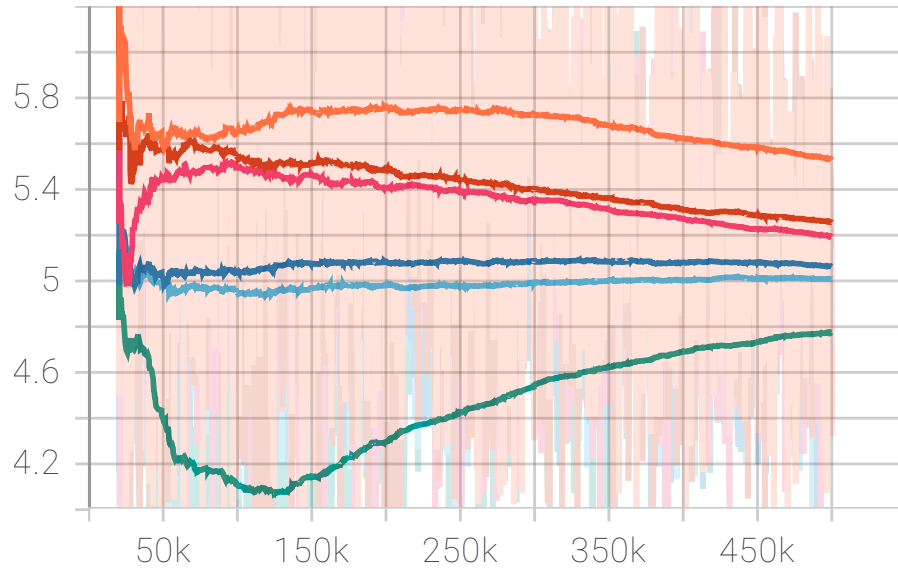


Figure 1: The best results — from `/runs/Apr27_23-01-58_andrew-XPS-15-9570`. The lines show the square root of the mean square losses with (a) $\text{lr}=0.3$ Alice (orange), Bob (dark blue); (b) $\text{lr}=0.1$ Alice (brick red), Bob (cyan); (c) $\text{lr}=0.01$ Alice (pink), Bob (green). The plot is from TensorFlow and uses smoothing of 0.999.

2 Results

2.1 Original strategies

Figure 1 is representative of the better results for the original strategies, **one per bit** — in other words, not very good.¹ Increasing from `h.GAMESIZE = 1` to `h.GAMESIZE = 32` gives no better results.

References

- [1] J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng et al., *Deep reinforcement learning with a natural language action space*, *arXiv preprint arXiv:1511.04636* (2015) .
- [2] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt et al., *Deep reinforcement learning in large discrete action spaces*, *arXiv preprint arXiv:1512.07679* (2015) .
- [3] S. J. Majeed and M. Hutter, *Exact reduction of huge action spaces in general reinforcement learning*, *arXiv preprint arXiv:2012.10200* (2020) .

¹The plot is taken from TensorBoard which gives an .svg file, then converted to .pdf by `rsvg-convert -f pdf -o <fig-file-name>.pdf "Sqrt losses.svg"`.