GuessTuples Project

Andrew J. Wren

3 May 2021

Abstract

Notes on `GuessTuples` project aka `NLearn`

1 Configuring the nets

1.1 Alice

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

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

$$r_{jk} \leftarrow r_{jk} - \frac{r_{j\checkmark} + (N_{\text{code}} - 1)\, r_{j\times}}{N_{\text{code}}}. \tag{1}$$

The Q estimate is then taken to be

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

where

$$b_j = \operatorname{sgn}(y_j) \tag{3}$$

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

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

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 $2^{N_{code}}$ is quite large... not impossibly so if $N_{code} = 8$.*

4. Inspired by Ref. [1], feed $\boldsymbol{x}$ into Alice's 'first' net, to get output $\boldsymbol{y}$, and all possible codes $\boldsymbol{c}$ into her 'second' net, both net's having the same target dimensionality (a hyperparameter). Then the code to use is the one $\boldsymbol{c}(\boldsymbol{x})$ closest to the output of the first net, with the Q being given by the inner product $Q = \langle \boldsymbol{y}, \boldsymbol{c}(\boldsymbol{x}) \rangle$. Ref. [2] might provide an alternative, actor–critic, approach on a similar theme. The main case above is, in effect, an embedding of $\boldsymbol{x}$ into the target space (of dimensionality N_{code}) which then compares with the natural embedding of $\boldsymbol{c}$ 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 $\boldsymbol{c}$ and Q-estimate for that code.

6. Move away from typical Q-learning. Instead Alice's output is the code $\boldsymbol{c}$ and then when Bob makes his choice $\boldsymbol{x}_{\text{pred}}$ (see below) run that choice through a copy of Alice, to get $\boldsymbol{c}_{\text{Bob}}$ and then the loss function is
$$L = -\,r(\boldsymbol{c}, \boldsymbol{c}_{\text{Bob}}). \tag{5}$$

1.2 Bob

One per bit aka **Simple.** Bob receives a matrix, $\mathbf{X} = (\boldsymbol{X}_i) = (X_{ij})$ for $0 \le i < N_{\text{select}},\ 0 \le j < N_{\text{elements}}$, and a code $\boldsymbol{c} = (c_k)_{k=0,\ldots,N_{\text{code}}}$. Why not makes his outputs be Q-estimates $\boldsymbol{z} = (z_i)_{i=0,\ldots,N_{\text{select}}}$. Bob's prediction is then $\boldsymbol{x}_{\text{pred}} = \boldsymbol{X}_{i_{\text{pred}}}$ where
$$i_{\text{pred}} = \operatorname{argmax}_i(z_i). \tag{6}$$
The loss function is
$$L = \left|z_{i_{\text{pred}}} - r\right|^2. \tag{7}$$

How do we enforce covariance with respect to the order of $(\boldsymbol{X}_i)$?

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
$$\mu \sum_{\sigma} \left|z - \sigma^{-1}\left[z(\sigma[\mathbf{X}])\right]\right|^2 \tag{8}$$
for some fixed hyperparameter $\mu > 0$. Note this the term is still run backward through the original $\boldsymbol{x} \mapsto z$ 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 $\boldsymbol{c}$, 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.

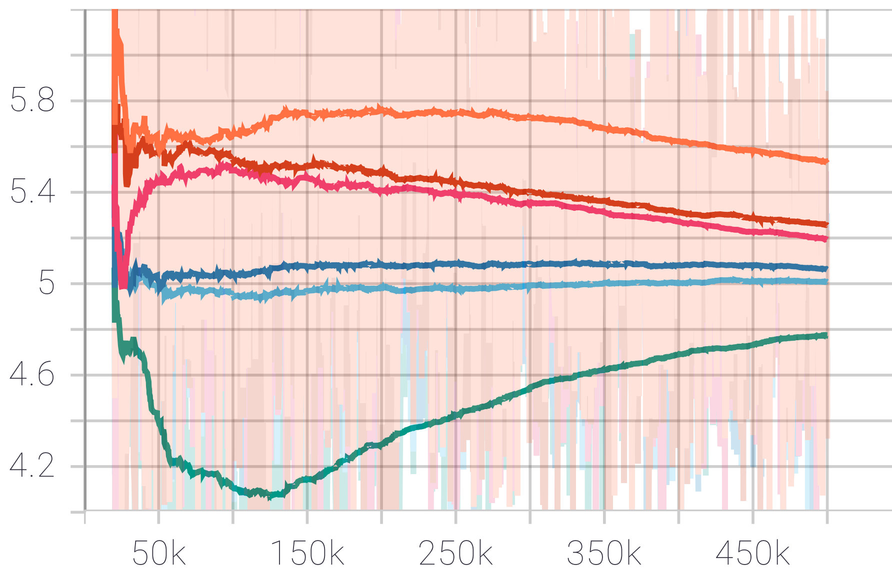

Figure 1: The best results — from `/runs/Apr27_23-01-58_andrew-XPS-15-9570`. The lines show the square root of the mean square losses with (a) `lr=0.3` Alice (orange), Bob (dark blue); (b) `lr=0.1` Alice (brick red), Bob (cyan); (c) `lr=0.01` Alice (pink), Bob (green). The plot is from TensorFlow and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

3 Revised approach — `NLearn`

Key runs:

1. `21-05-01_12:05:16` is the strategy that works

```
    'ALICE_STRATEGY': 'from_decisions',
    'BOB_STRATEGY': 'circular_vocab'
```

   up to a point when it levels off. Gets to `reward` = 0.6.

2. `21-05-01_20:04:35` other `lr` choices but same result — see Figure 2

3. `21-05-02_17:29:40` stops Alice training at some point. Alice `lr` = 0.1 and Bob `lr` = 0.01 gets to 0.8 — see Figure 3. The `config` includes

```
    hyperparameters = {
```

[1]The 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"`.

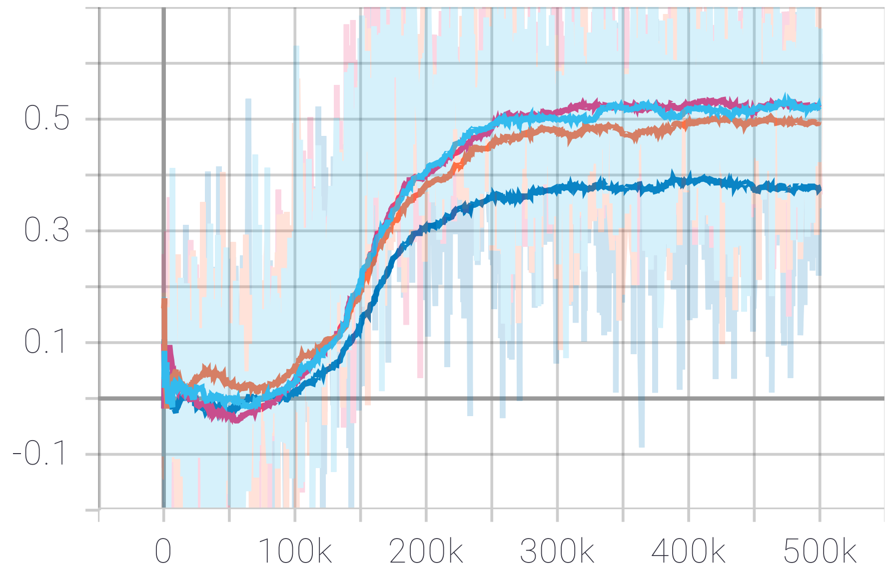

Figure 2: Mean Rewards per game for `21-05-01_20:04:35`. By colour, (Alice `lr`, Bob `lr`) are: cyan $(0.1, 0.1)$, orange $(0.1, 0.01)$, pink $(0.01, 0.1)$, and blue $(0.01, 0.01)$.

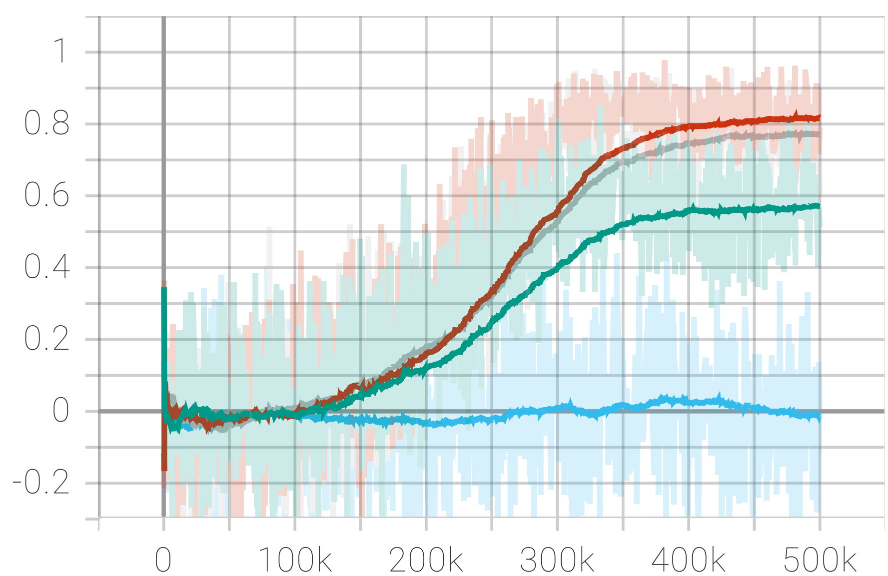

Figure 3: Mean Rewards per game for `21-05-02_17:29:40`. By colour, (Alice `lr`, Bob `lr`) are: green $(0.1, 0.1)$, orange $(0.1, 0.01)$, grey $(0.01, 0.1)$, and cyan $(0.01, 0.01)$.

```
        'N_ITERATIONS': 500000,
        'RANDOM_SEED': 42,
        'TORCH_RANDOM_SEED': 4242,
        'ALICE_LAYERS': 3,
        'ALICE_WIDTH': 50,
        'BOB_LAYERS': 3,
        'BOB_WIDTH': 50,
        'BATCHSIZE': 32,
        'GAMESIZE': 32,
        'BUFFER_CAPACITY': 640000,
        'START_TRAINING': 20000,
        'N_SELECT': 5,
        'EPSILON_ONE_END': 40000,
        'EPSILON_MIN': 0.01,
        'EPSILON_MIN_POINT': 300000,
        'ALICE_STRATEGY': 'from_decisions',
        'BOB_STRATEGY': 'circular_vocab',
        'ALICE_OPTIMIZER': ('SGD', '{"lr": 0.1}'),
        'BOB_OPTIMIZER': ('SGD', '{"lr": 0.01}'),
        'ALICE_LOSS_FUNCTION': ('MSE', {}),
        'BOB_LOSS_FUNCTION': 'Same',
        'ALICE_LAST_TRAINING': 200000
```

4. If increase `N_SELECT` to 16 (all the numbers shown to Bob), then, in run `21-05-03_10:53:10`, gets to `reward` = 0.8, as good as for `N_SELECT` = 5. Alice's code book is still very small:

```
    21-05-03_10:53:10BST_NLearn_model_1_Alice_iter500000

    01111010 [0, 15]
    01111100 [1, 2, 3, 4, 5, 14]
    01011100 [6, 7]
    01011110 [8, 9, 12, 13]
    01111110 [10, 11]
```

Things to try:

1. What codes does best Alice generate?
2. Try using the loss function to constraint outputs to nearer bit values.
3. How quickly can `epsilon` be tapered?
4. Vary learning rates.
5. Vary `modulus`, `N_CODE` and `N_SELECT`.
6. Introduce noise.

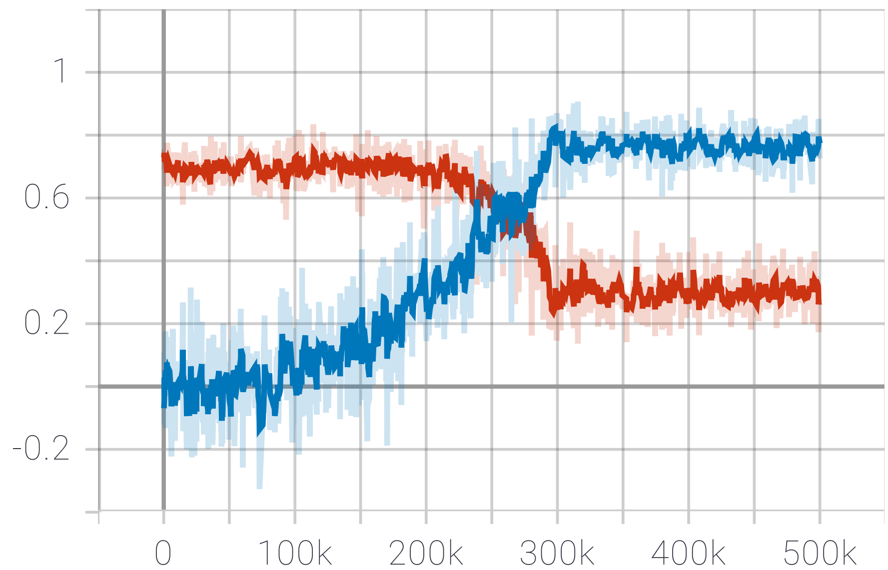

Figure 4: With `N_SELECT` = 16, at `21-05-03_10:53:10`.

7. Alice strategy with a code, as input and the output are values for the numbers. In each play (or train?) step feed all the codes in and the outputs indicate how well represents each number???
8. Try best strategy but with Alice outputs having dimension `2 ** N_CODE`.
9. Train bits successively.
10. Look at MARL literature.

The best Alice so far, `21-05-02_17:29:40 hp_run=2` generates codes as follows:

```
11101100 [0, 1, 2, 12, 13, 14, 15]
11101110 [3]
10101110 [4, 6]
10100110 [5, 7]
10100100 [8]
11100100 [9, 10, 11]
```

Surprisingly only six distinct codes used! At least the first and last have sequential runs of numbers.

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) .