GuessTuples Project

Andrew J. Wren

22 April 2021

Abstract

Notes on GuessTuples project

1 Configuring the nets

1.1 Alice

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 outputs for each possible code. *Might work but* $2^{N_{code}}$ *is quite large... not impossibly so if* $N_{code} = 10$.

1.2 Bob

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

$$i_{\text{pred}} = \operatorname{argmax}_i(z_i). \tag{5}$$

The loss function is

$$L = \left|z_{i_{\text{pred}}} - r\right|^2. \tag{6}$$

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

   $$\mu \sum_{\sigma} \left|\boldsymbol{z} - \sigma^{-1}\left[\boldsymbol{z}(\sigma[\mathbf{X}])\right]\right|^2 \tag{7}$$

   for some fixed hyperparameter $\mu > 0$. Note this the term is still run backward through the original $\boldsymbol{x} \mapsto \boldsymbol{z}$ 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. 6. *Seems the most straightforward?* **Adopt this for now.**

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.