MIMO detection problem using Deep Q-Network

Zeyu Li
Department of Computer Science
McGill University
Montreal, H3A 0B9
zeyu.li2@mail.mcgill.ca

Xinyu Wang
Department of Computer Science
McGill University
Montreal, H3A 0B9
xinyu.wang5@mail.mcgill.ca

Yichen Huang
Department of Computer Science
McGill University
Montreal, H3A 0B9
yichen.huang@mail.mcgill.ca

Abstract

In this experiment, we tried to solve a MIMO detection problem using a Deep Q-Network (DQN) approach, based on professor Jinming Wen and Xiao-Wen Chang's previous researches. The main goal is to find the best permutation of transmit antennas that maximizes the success probability of correctly detecting the transmitted symbols.

1 Introduction and Background

In contemporary communication channels, Multiple Input Multiple Output (MIMO) systems have gained considerable prominence, covering aspects such as time and frequency resources, multiple users and antennas. These systems yield substantial performance enhancements while simultaneously posing intricate detection challenges with high computational requirements. The field of deep machine learning is undergoing a significant evolution in recent years, paving the way for novel solutions to tackle the obstacles associated with MIMO systems[1]. In our study case, we derive inspiration from the Neural Sort algorithm, which focuses on decreasing computational resources. From prior researches and studies, a Convolutional Neural Network (CNN) was used to find and decide the optimal permutation. In contrast, from our point of view for this project, we have utilized a Deep Q-Network (DQN) approach to predict the optimal permutation matrix.

2 Problem Statement and Objective

Consider a MIMO system with N transmit antennas and k receive antennas. The received signal vector, $y \in \mathbb{R}^N$, can be represented as:

$$y = Hx + w$$

where $H \in \mathbb{R}^{N \times K}$ is the channel matrix, $x \in \mathbb{R}^K$ is the transmitted data symbol vector, and $w \in \mathbb{R}^N$ is the noise vector, which is a zero mean Gaussian variable of variance σ^2. We assume perfect Channel State Information (CSI), which means the channel matrix H is known exactly.

The object is to minimize the difference of the predicted transmitted data $\hat{x}$ and the original transmitted data x. In this project, we use success probability to estimate the likelihood of correctly recovering the transmitted data x from the given signal vector y and channel matrix H. The reward is the improvement of success probability after an action (a permutation), which is represented by:

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

$r = 10 \times (success\ probability\ after\ permutation - success\ probability\ before\ permutation)$

The purpose of applying a multiplication by 10 is to speed up the convergence. The valid action is applying one of all possible permutations of the given matrix.

The Babai estimator which has been discussed in previous studies is a method used to approximate the closest lattice point to a given point in a lattice. This problem, called the Closest Vector Problem (CVP), is a fundamental computational problem in lattice theory and has applications in areas such as cryptography, coding theory, and communications. It is based on a rounding-off technique applied to the coordinates of the given point. It finds an approximation to the closest lattice point by solving a sequence of unimodular triangular systems using a technique called LLL (Lenstra-Lenstra-Lovász) lattice reduction. While the Babai estimator does not guarantee an optimal solution, it is computationally efficient and works well in practice for many instances of the CVP. The previous studies preliminary decided the the performance of the estimator under specific constraints and provides insights into its success probability.

3 Methodology

3.1 Data preprocessing

The channel matrix is generated randomly and is flatten as states in DQN. The transmitted signal x is generated in 4-QAM form. In M Quadrature Amplitude Modulation (M-QAM), each transmitted symbol can take one of M possible values. For a MIMO system with 't' transmit antennas, the transmitted symbol vector x can take M^t possible combinations. In this project, we have 4 transmit antennas and 4-QAM 1, -1, 3, -3, so indeed x has $4^4 = 16$ combinations:

$$x = [a_1, a_2, a_3, a_4],\ a_i \in \{1, -1, 3, -3\}$$

The Gaussian noise is calculated as following:

$$noise\ power = \frac{||Hx||^2}{n * 10^{\frac{snr\ dB}{10}}}$$
$$noise = \sqrt{noise\ power} \times \mathcal{N}(\mu, \sigma^2)[2]$$

We first use the above generator to produce 10000 pairs of H, x, y to feed "Q learning" part and store the $(s, a, r, s', done)$ tuples in the buffer. Then we get the data to feed our "Deep learning" part.

3.2 DQN architecture

We implemented torch.nn package to construct the DQN model. The input layer is defined by the number of possible permutations, which represents the number of actions. The input layer is connected to the first hidden layer using a fully connected nn.Linear layer.

The DQN architecture consists of four hidden layers. For every layer other than the final hidden one, they will be followed by a ReLU activation function. The output layer is a fully connected layer with output size neurons, which represents the number of possible actions the agent can take. In this project, the output size is calculated as the factorial of the column numbers of the channel matrix, also known as the number of possible permutations.

3.3 Q learning features

The success probability is calculated using the QR decomposition of the permuted channel matrix H, and the reward is based on the difference between the action's success probability and the original success probability. Here is how we decide the success probability:

$$c_i^{\mathrm{BB}} = \left(\tilde{y}_i - \sum_{j=i+1}^{n} r_{ij} x_j^{\mathrm{BB}} \right) / r_{ii},$$

$$x_i^{\mathrm{BB}} = \begin{cases} \ell_i, & \text{if } \left\lfloor c_i^{\mathrm{BB}} \right\rceil \leq \ell_i \\ \left\lfloor c_i^{\mathrm{BB}} \right\rceil, & \text{if } \ell_i < \left\lfloor c_i^{\mathrm{BB}} \right\rceil < u_i \\ u_i, & \text{if } \left\lfloor c_i^{\mathrm{BB}} \right\rceil \geq u_i \end{cases}$$

$$P^{\mathrm{BB}} \equiv \Pr\left(\boldsymbol{x}^{\mathrm{BB}} = \hat{\boldsymbol{x}}\right)$$
$$= \prod_{i=1}^{n} \left[\frac{1}{u_i - \ell_i + 1} + \frac{u_i - \ell_i}{u_i - \ell_i + 1} \operatorname{erf}\left(\frac{r_{ii}}{2\sqrt{2}\sigma} \right) \right] \text{[3]}$$

In terms of the Q learning part, the states are the flatten matrices. The action space is all possible permutations of the input matrix. The reward r is $10 \times (success\ probability\ after\ permutation - success\ probability\ before\ permutation)$. The state transition is defined the same as the matrix permutation (but flatten matrix).

3.4 Training and validation process

The training function updates the DQN model using data from the buffer. It samples a batch of experiences, converts them to tensors, calculates current and target Q-values, computes the loss, and updates the model using backpropagation and the optimizer.

The agent will be trained for 4,000 episodes. The initial epsilon value and final epsilon valur for the epsilon-greedy exploration strategy are set to 1.0 and 0.1 respectively. The rate at which epsilon decreases, controlling the exploration-exploitation trade-off. The batch size is 32 and the discount factor γ is set to 0.99.

In terms of evaluation, we generate 1000 random MIMO systems, preprocess their channel matrices, obtain the training data in the buffer, and use the DQN model to predict the best permutation action. Then we count successful cases where the chosen action (permutation) improves the success probability. We consider an increase of success probability as a success case. The model's performance is represented by the ratio of successful cases to the total test number.

3.5 Hyperparameter tuning

When designing the DQN structure, considering the trade off between the computation complexity and the model performance, we chose a 5 layer DQN, which has relative good performance while still keeps the training time short enough. For other hyperparameters, we simply test different combinations of hyperparameter values to find the best-performing set. The approach is easy and simple to implement. However, it is very time costing and the optimal hyperparameter value might not be bounded by the interval we use. More details for optimization will be discussed in "Future Improvements" later on.

4 Results and Discussion

We use trained DQN to predict permutation from 1 to 9 times. After applying each number of permutations, we obtain that after applying the permutation given by the DQN model, the probability of the success probability of recovering transmitted symbol x increasing is 89.32% in average with standard deviation 0.84%. The original data is as following:

Permutation time	Performance (%)
1	89.90
2	90.10
3	91.80
4	90.10
5	90.80
6	89.80
7	90.70
8	90.90
9	89.80

Table 1: Performance results for different runs

5 Conclusion

In this project, we explored the application of Deep Q-Networks (DQN) to predict the optimal permutation matrix in a MIMO system. Our approach aimed at improving the success probability of recovering the transmitted symbol vector x. We used a five-layer DQN model to balance computational complexity and performance. Our evaluation on 1000 random MIMO systems showed that our trained DQN model achieved an average success probability improvement of 89.32% with a standard deviation of 0.84%.

The results indicate that our DQN-based approach can effectively increase the success probability of recovering the transmitted data in MIMO systems. However, there are still some potential areas for improvement, such as exploring more advanced hyperparameter tuning methods and considering alternative DQN structures.

6 Future improvements

We have been discussed several ways that might cause any improvements in the performance of the algorithm. Here are a few of them that could be helpful with future development. Due to lack of time we haven't tried them all but some of them are still worth to talk about.

For the reward calculation, we are currently using the success probability minus the original success probability and times 10. In this way, the rewards could converge faster than expected, and you can directly see the results of the comparison or whether if we have any improvements. We have already think of another way that could make the reward look easier than it is now, which is to normalize the values to organize them between 0 and 1. The benefits would be that the differences between the rewards calculated are standardized, so that improvements could be compared with each other using the same scale. Processing normalized rewards may also provide more reasonable interpretation for the results to show in graphs and tables.

Changing the architecture may also affect the training result. We can see that from previous experiments for using only three layers, the performance ends with approximately 70 percent, much lower than the current five layer model. For more hidden layers added, the performance will have a significant increase, and indeed the time cost also varies with the performance. Here is a table for comparison for five layers and seven layers.

layers	trail1	trail2	trail3	trail4	trail5	trail6	trail7	trail8
3	57.80%	60.10%	60.90%	58.60%	51.50%	62.00%	57.00%	57.10%
5	82.00%	81.00%	81.80%	82.00%	80.10%	80.00%	80.20%	80.60%
7	96.10%	95.50%	95.60%	96.10%	96.00%	95.30%	95.70%	96.30%

Another thing is that the termination condition we are currently using is to calculate the value of reward minus the reward from last episode. If the value is smaller than 0, which means there is no increase in the reward, them we would end the episode and identify the results. In this way, the whole training process terminates when there seems to be no more adjustment or optimization. However, this cannot discriminate whether the rewards are large enough or only a drop in the bucket. A more reasonable condition could be introduced to further more ensure the learning efficiency and accuracy, such as setting a threshold for the algorithm to accept the action step when the rewards are considered to be significant enough.

References

[1] S. Neev, D. Tzvi, and W. Ami. Deep mimo detection. In *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2017.

[2] J. G. Proakis and M. Salehi. *Digital communications*. McGraw-Hill., 2008.

[3] J. Wen and X.-W. Chang. Success probability of the babai estimators for box-constrained integer linear models. *IEEE Transactions on Information Theory*, 63(1):631–648, 2016.