# Bi-matrix Game Human Behavior Forecast Contest Writeup

## 1 Intro

To predict the probability distributions of row player actions given a game (and by extension the most popular action), we use a deep learning approach inspired by computer vision methods and learning a visual understanding of the game. Each game is presented to a human in a bimatrix format of row and column payoffs. Humans make their decisions based off this visual information, e.g. by scanning all row-column combinations and picking the most appealing action. Accordingly, we train a multi-layer convolutional neural network (CNN) to predict the frequencies associated with a game, treating the problem as a supervised regression problem. While the matrix format of the game payoffs makes them a convenient input for our CNNs, we add additional channels to our dataset as a form of feature engineering to add additional "economic priors", and show empirically that this leads to a boost in performance. We augment the given training data with payoff-invariant feature transformations to help train our models. Finally, although our deep learning approach is inspired by a potential human understanding of presented games and subsequent decision-making, we acknowledge that this approach is potentially orthogonal to how decisions are actually made; our model's discoveries may be quite different from how a human actually reasons. Nevertheless we try to shed some insight into how our models processes the given matrix data through Gradient-weighted Class Activation Mapping (Grad-CAM) [1], a technique that ordinarily highlights the particularly salient pixels of an input image for a trained image classification model. We use Grad-CAM to highlight the most important payoff squares to our models, potentially providing further leads to investigate for how a human processes a bimatrix game. Code for our implementation will be made available at https://github.com/mzio/cs236-pset1.

## 2 Approach

### 2.1 Data Processing and Augmentation

Although the bimatrix nature of the games make their payoffs a natural input to a CNN, we transform our data in several ways to lend further useful information to our models. In analogy to processing an image−where the red, blue, and green channels all lend complementary information to the model's visual understanding−we generate new features from the input row and column matrices, and include these as additional channels. We refer to the three additional channels added as **row_max_diff**, **col_max_diff**, and **row_col_diff**. The former two involve subtracting each value in a row or column player's payoff matrix by the max value. This is a simple way to get around the problem mentioned in [2], where individual elements of an input matrix are processed independently of each other, who instead proposed a new pooling layer. The **row_col_diff** channel involves subtracting each element's column payoff from the row payoff in a given game matrix. This captures information related to behavioral intuitions mentioned in previous work [3], where an action's appeal may be dependent on how much better the player's rewards are compared to the counter-party's. Cooperative players may try to select an action that minimizes the value in this channel while maximizing the value in the other channels (i.e. both players do well), while others may choose to just pick their best-looking payoff. We hypothesize that having these values as direct inputs to our model could be advantageous.

To compare the advantages of these additional channels we create several datasets that each include a subset of these channels (Table 1), and compared model performance after training on each of these.

Finally, as the success of supervised deep learning approaches is heavily contingent on the training dataset size, we augment our data by performing fixed-payoff transformations. We do this by swapping the row payoff actions with each other, and adjusting the action frequencies accordingly. For example, if action 1 gives payoffs $A, B, C$ and action 2 gives payoffs $X, Y, Z$, and the frequencies for 1 and 2 are 0.6 and 0.3, then we can generate a counterpart where action 1's payoffs are $X, Y, Z$ and action 2's payoffs

Table 1: Generated training datasets

| Dataset | Channels Included |
|---|---|
| Payoff | row_payoffs, column_payoffs |
| Diff | row_col_diff, row_max_diff, col_max_diff |
| All | row_payoffs, column_payoffs, row_col_diff, row_max_diff, col_max_diff |
| Row | row_payoffs, row_col_diff, row_max_diff |

are $A, B, C$, with frequencies $0.3, 0.6$ for actions 1 and 2 respectively. This approach is similar to how image classification datasets can be augmented by rotating the images, which changes the input values but preserves the actual class. We justify this augmentation by assuming that players pick actions regardless of where the payoffs are physically located within a matrix. With $3 \times 3$ bimatrix games, we go from our provided dataset of 250 games to 1500 games.

## 2.2 Bimatrix Game CNN (BimatrixGameNet)

We train a multi-layer CNN to predict the frequencies of the row player's actions given an input game. Because the input is rather small, we use padding to maintain fixed-size feature maps with $2 \times 2$ kernels, and refrain from any pooling layers. The last layer of our CNN outputs three values corresponding to each action's frequencies, which we softmax to maintain a probability distribution. We then picked the predicted most frequent action based on which action had the largest predicted probability. We use ReLU for all other layers, and employ dropout = 0.5 for the fully-connected layers. An overview of our model architecture can be found in Figure 1.

To train our model we considered both a mean squared error (MSE) loss with respect to each action frequency, and a KL divergence loss against the entire output vector. To compare different models trained on the different datasets in Table 1, we separated our augmented datasets into train, validation, and test splits composed of 0.6, 0.2, and 0.2 of the data respectively. We trained with the Adam optimizer setting learning rate $= 1e^{-4}$ and batch size 4. We performed a small architecture search and tuned hyperparameters for each model by training on the train and evaluating on the validation set. We then selected the optimal hyperparameters for each to train on the combined train and validation sets, then compared the quadratic losses and accuracy metrics produced by evaluating on the test set. For our submission we selected the best performing model and trained on the entire provided training data, and submitted the predictions when performing inference on the provided test data.

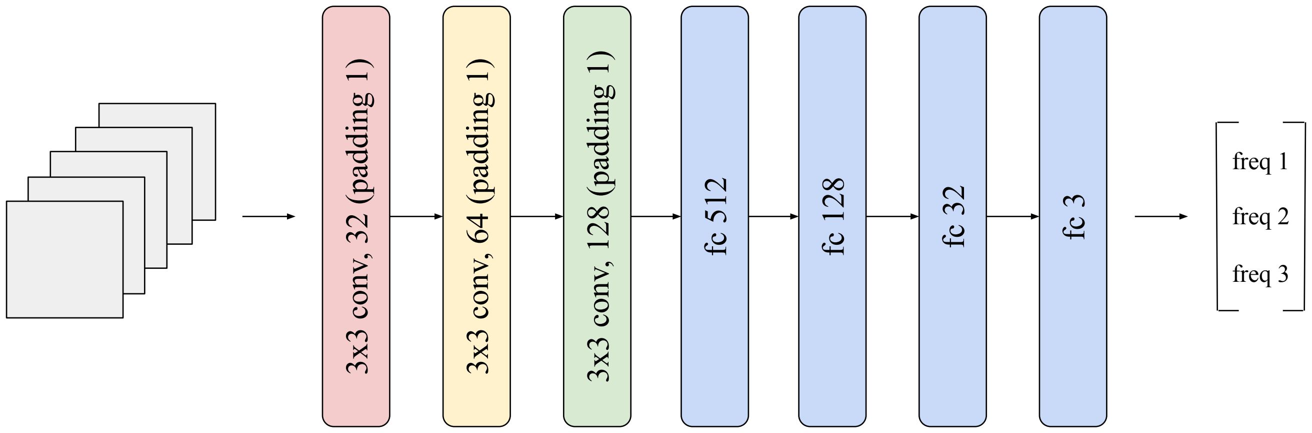


Figure 1: Overview of our CNN architecture. We feed in our generated multi-channel input from the game data and output a vector of probabilities pertaining to each action's frequency for that game. Each layer is connected by a ReLU activation layer. Additionally to stablize training and prevent overfitting we employ dropout = 0.1 for our convolutional layers (conv) and dropout = 0.5 for our fully connected (fc) layers.

## 3 Results

Our results evaluating on the test sets of our training data are summarized in Table 2. We report numbers after training with 200 epochs, and the summed quadratic distance over all games in the table. We found that training with MSE loss did better than KL divergence across all datasets, and ultimately the model trained on the "All" dataset performed the best. For performance on the entire provided training data we achieved a mean quadratic loss of **0.0533** and accuracy of **80.467**%. Because we trained our model on this data, we calculated these numbers by averaging the test split results from a 5-fold cross-validation. Accordingly these results are different from our best value in Table 2, evaluated on a separate test split.

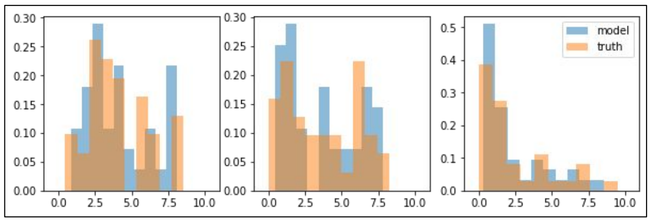
Table 2: Bimatrix Game Prediction Results on Testing Subset

| Dataset | Loss Function | Quadratic Distance | Accuracy (%) |
|---|---|---|---|
| Payoff | KL | 22.435 | 72.0 |
| Payoff | MSE | 21.364 | 73.0 |
| Diff | KL | 19.794 | 77.0 |
| Diff | MSE | 18.091 | 77.3 |
| Row | KL | 20.069 | 71.7 |
| Row | MSE | 17.019 | 77 |
| All | KL | 19.178 | 73.3 |
| **All** | **MSE** | **16.629** | **78.0** |

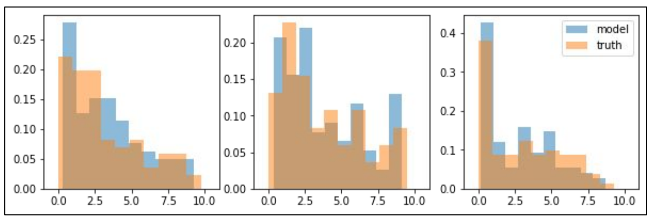
## 4 Discoveries

### 4.1 Game-specific Action Frequencies

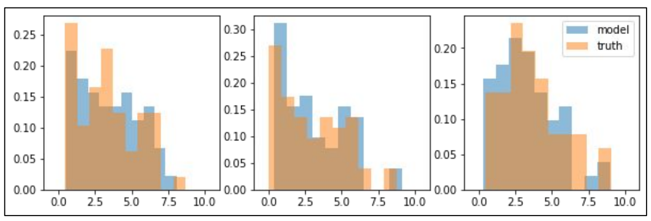
As previous work has shown that model performance can vary based on the type of games, such as in [3] where their decision tree performed worse on the algorithm-generated games compared to the lab games, we wondered how robust our model was to different "types" of games. To explore this we organized the given training data's games into clusters by our channels' characteristics using K-means clustering, and then plotted both our model's predicted and the groundtruth action distributions in each discovered cluster (Figure 2). We found that across clusters, our predicted distributions seem to align with the groundtruths, although specific actions in specific clusters are predicted better than others. This suggests our model is able to generalize across different types of games.



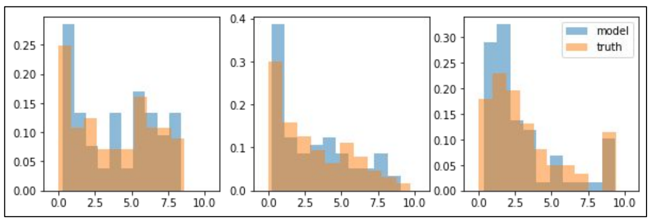
(a) Game Cluster 1



(b) Game Cluster 2



(c) Game Cluster 3



(d) Game Cluster 4

Figure 2: Predicted and groundtruth frequencies of player actions in bimatrix games. We visualize the distribution of action frequencies ([left, center, right] in each box denotes actions 1, 2, and 3; frequency multiplied by 10 to give a density) across four game clusters in the provided training data discovered through K-means clustering. Across each cluster our model predicts a similar action distribution compared to the groundtruth.

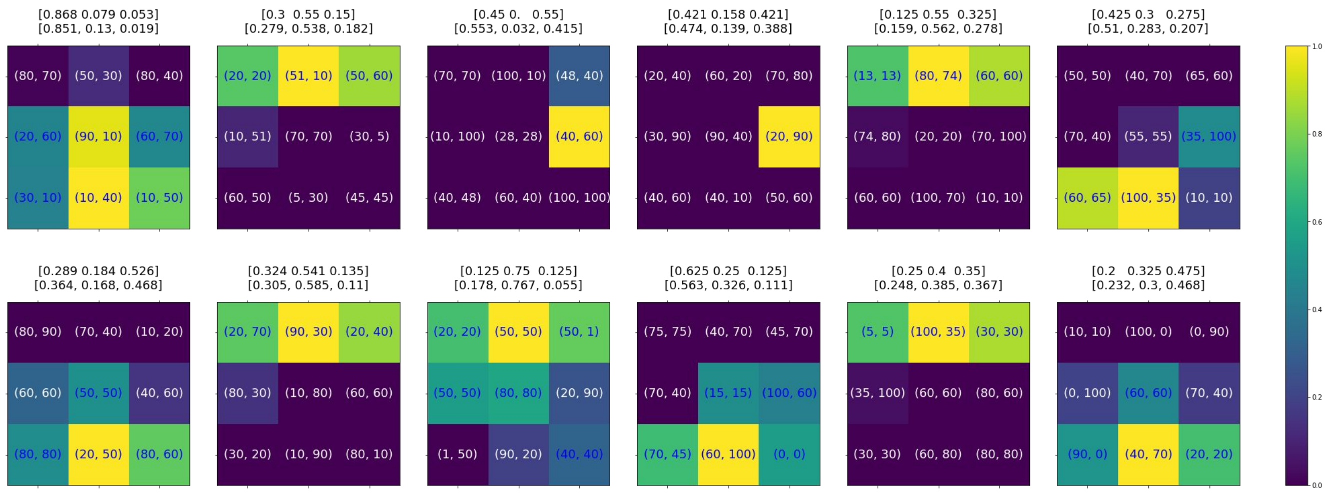


Figure 3: Gradient Class Activation Maps showing spatial attention to input game payoffs. Each matrix depicts a given bimatrix game, with row and column payoffs for that cell shown. Above is the groundtruth frequency of actions (1, 2, 3) (top) and the prediction action (below). Lighter means more attention was paid to that cell.

### 4.2 Grad-CAM Spatial Attention

Finally in an attempt to visually explain how our model arrived at its frequency distributions, we use Grad-CAM to visualize the learned "attention" paid to each payoff cell. More details regarding the algorithm can be found in [1], but briefly Grad-CAM works by using the gradients of the target layer (our final softmax) with respect to the final convolutional layer to produce a localization map that highlights important regions in the input image based on the final convolutional layer's activations. Intuitively it identifies the parts of an input that provide the strongest signals for predicting the final output.

In Figure 3 we depict visualizations using our best performing model for the first 12 games in the training set. Although from a quick inspection we do not see any obvious patterns across all examples with regard to the most salient payoff cells, perhaps further analysis is required to tease out the significance of the highlighted areas. One important thing to note is that the most attended to cells do not seem to correspond to the row representing the action most highly predicted. In fact, in cases where the difference in frequencies of actions is drastic (e.g. top leftmost or top center-left and center-right), the highlighted cells correspond to less frequently chosen action-rows, and the most frequent rows seem rather unattended to. We do note that among the attention maps, similar maps correspond to similar predicted distributions (e.g. top and bottom second leftmost, or bottom right and leftmost), which is expected due to how we calculate the attention mappings.

## References

[1] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *CoRR*, abs/1610.02391, 2016.

[2] Jason S Hartford, James R Wright, and Kevin Leyton-Brown. Deep learning for predicting human strategic behavior. In *Advances in Neural Information Processing Systems*, pages 2424–2432, 2016.

[3] Drew Fudenberg and Annie Liang. Predicting and understanding initial play. *American Economic Review*, 109(12):4112–41, 2019.