MyoshuGo - Review-Board Matching for Go Games

Weiqiu You **Rebecca Iglesias-Flores** **Di Wu** **Xiaoyue Sun**
University of Pennsylvania
{weiqiuy, irebecca, wudi930325}@seas.upenn.edu
sunxy30@wharton.upenn.edu

Abstract

Predicting the next move and win in go games has been achieved by massive self-plays in reinforcement learning, but even the best AI can only predict winrates without explanation. In this paper, we propose a new task of giving potential explanation of evaluation of game positions by predicting whether a board position and a review/comment match each other. Matching review will be able to provide an explanation for why the player's move is good or bad. We conduct experiments using logistic regression baseline, end-to-end neural networks models, and neural models combining pretrained features. Although we are far from solving the problem, we show that pretraining is essential to the problem and that multimodel interaction is required in model.

1 Introduction and Motivation

Game of Go has been a popular subject for deep learning research because the game entails simple rule but allows various tactics to be played during the game. With many of the recent research focused on predicting next move, we plan to apply deep learning from a different aspect of the game: move evaluation using natural language. Move evaluation involves estimating the current game landscaped or final outcome given latest move and board of the game.

It is a very hard task due to the nature of the task: situation evaluation and comments generation. Situation evaluation itself is a good research direction and it is not easy even for professional players. There are a lot of Go software companies(Tencent, Google, etc.) working on winning rate prediction AI which is a much easier goal compared to what we do and even they sometime fail to achieve a great accuracy. In terms of the comment generation, GAN and GPT-3 works well on generating one type of sentences but it is hard to incorporate it within our purpose. The comments of the move is not only just good or bad but also including the risk and analysis of potential benefit. As we discussed, to generate a comments generator is long way to go. To make it more realistic, we simplify the problem into a classification problem.

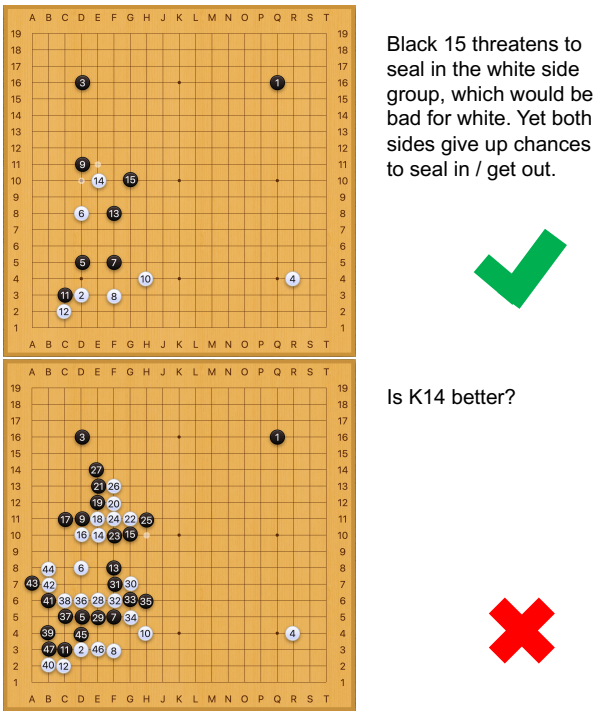


Figure 1: The first board matches with the review comment, as black is actually surrounding the two white stones and making it hard for white. But the second comment does not match the second board as K14 is not related to the current battle.

In this project, we try to evaluate each move by

matching it with the correct comment made by a commentator. Given a board position and a comment, the model needs to decide whether they match or not, as shown in Figure 1. The comment usually evaluates how good or bad the move is and how it might influence the following play. Matching the reviews is slightly different from the traditional position evaluation where winning chances are calculated. Here we focused more on player's intention behind each movement and potential good or bad decisions they have made that worth pointing out.

The rest of the report is organized as the follows. Section 2 reviews existing working and literature. Section 3 provides a brief discussion of the data and the problem. Section 4, 5, 6, and 7 lays out the model we implemented. We then move on to presents and analyze the experimental results in section 8. Section 9 concludes this reports.

2 Related Work

Our research problem is similar to an image and text matching problem. Thus we need appropriate models to represent image and text.

The most common representation of go boards is Convolutional Neural Networks (CNN) (Silver et al., 2016) which is widely used in image representation. Another possible architecture that we hope to explore is Graph Neural Network (GNN) (Wu et al., 2020) which makes sense to use for boards that are 2-dimensional. Recently, as unified image and text representations become more and more popular, it is also potentially possible modify Transformer (Vaswani et al., 2017; Li et al., 2019; Lu et al., 2019) for board representation. As for pretrained board representations, ones that are trained on predicting the winrate of currect board position and to play the game could be used. KataGo (Wu, 2020) builds on top of AlphaZero (Silver et al., 2017) by adding go specific features as input, whose intermediate representation can be used for our project.

For text representations, there are basic bag-of-words model(Harris, 1954) that represent each word as the words in its context window, recurrent neural networks (Hochreiter and Schmidhuber, 1997), and self-attention models (Vaswani et al., 2017).

When deciding if a pair of image and text match, we can use some functions to compute the similarity scores (Zhang et al., 2020), including concatenation, element-wise product, or going through a nonlinear transformation.

3 Problem Formulation

3.1 Problem Definition

The problem we are trying to solve is commenting Go moves. Given the situation S_i and the current move m_i, our model is trying to provide a basic analysis and give the correct review about this move, like if it's a tesuji(good move) or it causes a bad consequence. To be specific, we will provide the true review with another false review, which is generated from other moves and situations, to the model. Then the model needs to determine which one is correct. The loss function is shown as follow:

$$Loss(r_i, \hat{r_i}) = \begin{cases} 1, & \text{if } r_i = \hat{r_i} \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

where r_i is the correct review and $\hat{r_i}$ is predicted by our model $\hat{r_i} = f(m_i, S_i)$.

3.2 Data

Our data is the real Go scores with teachers' reviews downloaded from (Arno, 2016 (accessed November 16, 2020). For each score, since most of moves do not have comments, we only extract moves that having reviews along with its situations as our data. We treat each pair of board and comments as one sample. After random sample another negative review from the other moves, we will get our final dataset. In total, we have 10000 games. We divided the games into training, validation and test sets, with ratio of roughly 7:3 for training and testing, and 10% of training is used for validation. We filtered out games that are not using standard 19×19 board and only use the positions in the games which have comments. The number of positions used are shown in Table 1. As we have each board paired with one randomly selected negative text examples, we have a total of $2\times$ pairwise examples as from the table. [1]

We preprocess the data by tokenizing with moses-decoder (mos, 2020 (accessed November 16, 2020)

[1]The projected started out with using 9 negative text comments, we decided to change to 1 negative text comments to have more balanced data.

and applying byte pair encoding with fastBPE (Sennrich et al., 2016) to split words into subwords with 32000 operations. Parameters and text vocabulary are shown in Table 3.

Train	209566
Validation	20676
Test	91468

Table 1: Number of positions used in train, validation and test sets

Board	361
Current Move	3
Text Vocabulary	30116

Table 2: Parameters of board and text vocabulary

4 Logistic Regression Baseline

Two sets of features will be leveraged in our prediction baseline model: board configurations (+ its associated moves) and reviews. The board features are generated separately using a Sklearn's DictVectorizer() and the review features are generated using Sklearn's CountVectorizer() (standard BoWs). Furthermore, a manual grid search is performed to explore varying settings for solvers, regularization values, and penalities and then compared to that of an automatic grid search using Sklearn's Grid Search to ensure the reliability and integrity of baseline performance. The hyperparameters verified during the grid search on the validation set will be used for the final baseline reported results on the test set.

Two baselines were explored: 1) Manually engineered features (see §4.1), and 2) Pretrained features from Katago GitHub Model for the board and Transformer encoder Model for the reviews (see §4.2). The pretrained features are the same ones that will be used in our sophisticated models (see §5 and §7), so I thought I would do one baseline with manually engineered features and one with the same pretrained features that the sophisticated models will be using.

4.1 Logistic Regression using manually engineered Features

The manually engineered features were generated by using Sklearns DictVectorizer() and contained 1) the move, 2) color of the stone playing that turn, and 3) the 19x19 board. The 3 different features were concatenated together and flattened for a final 364-dimensional representation of the board for each play, see §4.1.1 as reference. The review comments were manually engineered by using Sklearns Bag of Words CountVectorizer(). The review comments were added to the CounterVectorizer() and only a max of 1000 words were taken into account. The review comments resulted in a 1000-dimensional representation of text for each play.

The board features and text features for each board were concatentated together for a total of a (364+1000) 1364 dimensional representation for each example. For the positive examples the (true board representation + true review representation) were concatenated and for the negative example the (true board representation + the wrong review representation) were concatenated. For preliminary results, we used a 10/90 split positive to negative examples, which was later changed to a 50/50 split for the final results. Below is a summary of the datasets used for logistic regression baseline using the manually engineered features:

4.1.1 Datasets (constructed with manually engineered features

- Training, Validation, and Test examples

```
---TRAIN EXAMPLES---
board.shape: (209566, 364)
text.shape: (209566, 1000)
X, y: ((2095660, 1364), (2095660,))
Time elapsed making boards: 00:04:14
Time elapsed making text: 00:00:10
Time getting examples:  00:00:33

---VALIDATION EXAMPLES---
board.shape: (20676, 364)
text.shape: (20676, 1000)
X, y: ((206760, 1364), (206760,))
Time elapsed making boards: 00:00:24
Time elapsed making text: 00:00:01
Time getting examples:  00:00:02

---TEST EXAMPLES---
board.shape: (91468, 364)
Time elapsed making text: 00:00:04
X, y: ((914680, 1364), (914680,))
Time elapsed making boards: 00:01:47
```

```
text.shape: (91468, 1000)
Time getting examples:  00:00:11
```

Listing 1: Dataset split using manually engineered features

4.2 Logistic Regression using Pretrained Features

The pretrained features were generated by using an alphaGo Katago Pretrained Model on GitHub ((Wu, 2019) which has the ability to extract a myriad of features, but for our intents and purposes we use the 'trunk' feature of the model. The 'trunk' feature returns a layer from the model that has gone through several of convolutional layers describing every aspect of the game, so the 'trunk' embodies the full representation of the board at its best. The pretrained board representation is of dimension (19,19,128) and then is flatten to (361,128) for each play. The pretrained review comments were generated by a pretrained transformer encoder, with dimension (100,200). The review comments were truncated to (100,128) to match the last dimension of the board, so the dot product could be taken of these later.

The outer dot product of the board features and text features were taken for a total of a (361,128)*(128,100) = (361,100) combination representation of the text and board. The (361,100) dimensional vector was flattened out for a final 36100-dimensional pretrained board/feature combination representation. Finally we used PCA to reduce dimensions to 100 components (we did this for faster processing). The final representation using the pretrained features is (nexamples, 100). For the positive examples the (true board representation + true review representation) were dot-producted together and for the negative example the (true board representation + the wrong review representation) were dot-producted together. There was no autogrid parameter search done using the pretrained features and therefore only preliminary results were reported using the manually generated features. The logistic regression baseline using the pretrained features was done on the best forming parameters found on the hyperparameters found running it on the manually engineered features. This was due to a time constraint. Below is a summary of the datasets used for logistic regression baseline using the pretrained features:

5 Neural Model With Generated Board and Text Features

With the go board and steps of the moves, one can apply a network model to the board feature to represent given board and steps of the play. Then for the comments/review of the current play we will rely on the LSTM to generate the representation of matching comments. Since we have two sets of very different inputs, we will explore different neural net structures for the two different set of features. We started with applying convolutional and linear layer for the board and moves. For the text, we first load the text uses torch text and vectorized the input text used pre-build vocabulary. We constructed an embedding layer, followed by a bi-LSTM layers, and connected with a fully connected layer.

Model Construction We have explored different model structures to handle the generated board and text features. We can generalized the model structure for the go board into two different general categories.

1. Fully connected neural networks: The input structures for board and steps has fairly simple structures, apply a series of fully connected layers requires no additional assumptions of the input data. However, fully connected networks is prone to be over-fitting.

2. Convolutional neural networks: the neural network we presented here is a fairly simple networks. Since here, we try to learn a 2D patterns of the board. We could apply convolutional layers with ReLU activation function

Model Architecture Table 3 summarizes the general model architectures we used to trian the neural model with generated board and texts. We have experiments with convolutional layers with different padding and stride, we also add additional fully connected layer on top of three layers structures listed below. We also have experiments with other activation functions including $Tanh()$ or $Softplus()$

6 Pretraining

6.1 Board Embedding Pretraining

We get pretrained board embedding from a pretrained KataGo model (Wu, 2020). KataGo extends AlphaZero (Silver et al., 2017) by revisiting traditional

Board	Text
Conv Layer 1 ReLU 1 Conv Layer 2 ReLU 2 Conv Layer 3	Bi-Directional LSTM with variable length inputs
Fully Connected Layer 1	
ReLU 3	
Dropout 1	
Fully Connected Layer 2	
ReLU 4	
Dropout 2	
Fully Connected Layer 3	

Table 3: Neural Model Structures

Go playing programs and adding go specific features. KataGo consists of a stack of residual blocks that contains batch-normalization layers, ReLU activation functions and convolutional layers.Two blocks also contain a global pooling bias structure. After the trunk, there is a policy head and a valuehead.

We utilize an intermediate layer right before the policy and value heads. For efficiency, we use the 10 block model recommended for fast use by KataGo. The intermediate layer extracted has a dimension of (19, 19, 128) for each board.

6.2 Text Embedding Pretraining

The text embedding borrows the idea from autoencoder architecture with transformer (Vaswani et al., 2017). Autoencoder is a type of unsupervised neural network to learn efficient data coding. By using the data itself as the "label", autoencoder can utilize almost unlimited cheap data to pretrain the model. Transformer is the state-of-art language model to deal with text data. It overcomes the long-term memory and reading direction issues, which were two of the most important challenges in NLP domain, by Masked Multi-Head Attention mechanism.

In our case, one of the challenges we meet is how to represent the comments. Since the length of comments varies from tens to thousands, it's easy to have a sparse and not informative sentence with a lot of paddings. In addition, our text data contains bunch of Go terminologies, which makes transfer learning or directly using trained model much harder. Moreover, limited by the time and computational resource, we can't afford ahuge network to get the representation and later classification.

To solve above problems, we firstly truncate the comments by 100 tokens. It includes more than 95% full sentence among the data and significantly improves the training speed and output feature size. In addition,we introduce the transformer encoder to do the text embedding part. It consists of two layers of transforme ras encoder and two fully connected layers as decoders in the pretraining stage. To avoid overfitting, we adddropout layer between fully connected layers and shrink the dimensionality of the feature from encoder.

After pretraining, we remove the decoder part and load weights to extract features from text. Since our dataset is too big, it may need over 200GB if we simply dump all data and features to memory. Therefore,we convert the encoder part as a function so that it can extract features by mini-batch.

7 Neural Models with Pretrained Board and Text Features

We experiment with two different ways of combining pretrained board and text features in a neural model: concatenation and dot product. We planned to also experiment with multiheaded attention but due to limited time we had not time to train and hyperparameter search for this model so we will leave it for the future.

Concatenation The most easy to think of way to combine two sets of features of different sources is concatenation. In our case, for each batch of board and text pairs, we have board features of shape (batch size, 19, 19, 128) and text features of shape (batch size, 100, 200). We flatten out all the dimensions and have a single linear layer projecting from 19*19*128 + 100*200 to 1 dimension. This model is basically the same as logistic regression baseline except that it does not average the features but have weights for all the features.

Dot product As pointed out by Hessel and Lee (2020), there is the difference between multimodal additive models and multimodal multiplicative models. Multimodal additive models might not be able to solve problems that require interaction between

images and texts. In our case, we need to decide if a board and a piece of text match each other. Simply computing the score of each modality and adding together will likely not be able to do the job. Therefore, we first project board and text features to hidden dimension of 512 respectively. Then we take the dot product of the transformed board and text features.

After obtaining logits with either of the three methods, we use BCEWithLogitsLoss() from PyTorch which contains a softmax and binary cross entropy loss.

8 Experimental Results

8.1 Baseline

A preliminary baseline model was experimented on using logistic regression on full data (2M examples)and smaller subsets of data (20K examples and 100 examples). Reported here are the results on the small subsets, we encountered memory issues with the 2M full dataset and are currently exploring reasons for this divergence to overcome this at the next checkpoint. Additionally, a manual grid search vs. sklearn's internal GridSearchCV() method was carried out in order to compare results as a confidence measure and research integrity. The preliminary results are unexpected and suspect, but this was discussed at the first check-in meeting with Lyle and in the Conclusion and Discussion (seeÂ§6) we show what changes are going to be made in order to acquire better results for the next checkpoint. Finally, at the next checkpoint we will also include a hyperparameter search involve verbosity, solvers, and penalties. The following results are only doing a parameter search on various regularization values of C.

8.1.1 Preliminary results with Logistic Regression Baseline

- Regularization Values Searched (C): [0.001, 0.01, 0.1, 1.0]
  NOTE: The following results reflect the initial 10/90 split of positive to negative examples for training, except for the last one which is a 50/50 split.
  NOTE: The following results are reported on the validation set

- —Manual Grid Search for Baseline Logistic Regression—

# of examples	Best C	Acc
100 (10/90 split)	0.001	2%
20K (10/90 split)	0.001	14.3%
20K (50/50 split)	0.001	63.6

- —Auto Grid Search for Baseline Logistic Regression—

# of examples	Best C	Acc
100 (10/90 split)	0.001	1.5%
20K (10/90 split)	0.01	15%
20K (50/50 split)	0.01	57.4%

- Using 50/50 split final results for manual and auto grid search

```
---Results Manual Grid search---
train_best_params: (0, 0.001, 51.2%)

val_best_parameters: (0, 0.001,
    63.6%)
test_overall_accuracy:
    57.435387239253075

----Results Auto Grid search----
test best parameters: {'C': 0.1}
val best parameters: {'C': 0.1}
test_overall_accuracy:
    57.435387239253075
```

Listing 2: Comparison of Hyperparameter Search Between Manual and Auto

Some concerns that I have on this first run is that the best parameter value I get for C when running the manual search (the manual method I implemented) does not always match with what sklearn's internal autogrid search determines. This could be because I am only using 20 thousand examples and so there ismore noise to consider or abysmally underfit. It might be the case when I expand this to the full 2M example dataset that the best found hyperparameters will always match, but essentially this underscores why this sanity check is necessary for us to carry out. Furthermore, in listing 2, you can see that the final manual vs.auto search comparison on the 50/50 split shows different best hyperparameter values, but resulting in the same accuracy. Again, this could be because I am only using a small subset of the data for this checkpoint,and when I expand it later this will go away. To be determined.

8.1.2 Final results with Logistic Regression Baseline

AutoGrid Search using a 50/50 split (different from above where we initally had a 10/90 split). **I show time studies at the end of EACH search to show the substantial amount of time it took to find the hyperparameters right for our data (ranging from 1 hr - 15 hrs of processing time for each group)**.

- —Auto Grid Search L1 on 2K examples—

Metric	Best Model	Acc
Precision	C:1, penalty:l1, solver:liblinear	0.48%
Recall	C:1, penalty:l1, solver:liblinear	0.48%
F1-Score	C:1, penalty:l1, solver:liblinear	0.48%

Time elapsed: 00:07:26

- —Auto Grid Search L1 on 20K examples—

Metric	Best Model	Acc
Precision	C:10, penalty:l1, solver:liblinear	0.47%
Recall	C:10, penalty:l1, solver:liblinear	0.47%
F1-Score	C:10, penalty:l1, solver:liblinear	0.47%

Time elapsed: 00:08:25

- —Auto Grid Search L2 on 2K examples—

Metric	Best Model	Acc
Precision	C:10, penalty:l2, solver:sag	0.36%
Recall	C:10, penalty:l2, solver:sag	0.36%
F1-Score	C:10, penalty:l2, solver:sag	0.36%

Time elapsed: 01:08:00

- —Auto Grid Search L2 on 20K examples—

Metric	Best Model	Acc
Precision	C:10, penalty:l2, solver:liblinear	0.48%
Recall	C:10, penalty:l2, solver:liblinear	0.48%
F1-Score	C:10, penalty:l2, solver:liblinear	0.48%

Time elapsed: 15:20:55

- —Auto Grid Search ELASTIC on 2K examples—

Metric	Best Model	Acc
Precision	C:10, l1_ratio:0.25, penalty:elasticnet, solver:saga	0.35%
Recall	C:10, l1_ratio:0.25, penalty:elasticnet, solver:saga	0.35%
F1-Score	C:10, l1_ratio:0.25, penalty:elasticnet, solver:saga	0.35%

Time elapsed: 12:49:12

- —Auto Grid Search ELASTIC on 20K examples—

Metric	Best Model	Acc
Precision	C:10, l1_ratio:0.25, penalty:elasticnet, solver:saga	0.46%
Recall	C:10, l1_ratio:0.25, penalty:elasticnet, solver:saga	0.46%
F1-Score	C:10, l1_ratio:0.25, penalty:elasticnet, solver:saga	0.46%

Time elapsed in main: 07:50:17

- —Summary of Best Performing Comparison between L1,L2,Elastic—

Metric	Best Model	Acc
l1	C:10, penalty:l1, solver:liblinear	0.47%
l2	C:10, penalty:l2, solver:liblinear	0.48%
Elastic	C:10, l1_ratio:0.25, penalty:elasticnet, solver:saga	0.46%

Time elapsed in main: 07:50:17

```
Detailed classification report:


The model is trained on the full development set.
The scores are computed on the full evaluation set.


              precision    recall  f1-score   support

           0       0.49      0.87      0.63     91468
           1       0.43      0.10      0.16     91468

    accuracy                           0.48    182936
   macro avg       0.46      0.48      0.39    182936
weighted avg       0.46      0.48      0.39    182936
```

Figure 2: Detailed Classification Report from Autogrid Search on 20K using the best performing L2 parameters

- Test Accuracy on Full Dataset

Type	Best Model	Acc
Manual (50/50 split)	C:10, penalty:l2, solver:liblinear	0.50%
Pre-trained (50/50 split)	Pending	Pending

Test accuracy on the baseline using the pre-trained features was not able to be completed due to memory errors. These were the things I tried in order to solve the memory issue

- lazy loading all of the file we read in
- switching our files to database .h5 files (similar to SQLDictionaries) instead of keeping the in pickle.
- we tried switching to smaller models (there were two version of Katago and swapped it out for the smaller model to support memory issues)
- we tried batching the extraction of features from the katago model for faster processing (before it was extracting features for one board at a time and we changed source code to be able to support extracting the features in batches)
- reducing the amount of training examples we based our parameter search on
- two kinds of concatentation, taking the average and combining dimensions, dot product - all in an effort to try to reduce the dimensionality of the pretrained feature to be able to fit in memory
- we tried using lazy loading of batches in order to yeild generators that would later free up memory
- after variables were used, we tried zeroing them out so as to free that memory.
- we tried to use PCA to reduce the number of features

We learned a lot of these techniques during the process and had we known them up front, we could have better planned for the pretrained features and be able to have results, but it was too soon to the deadline by the time we learned all of this.

These were the hyperparameters searched in our autogrid search for the different penalties:

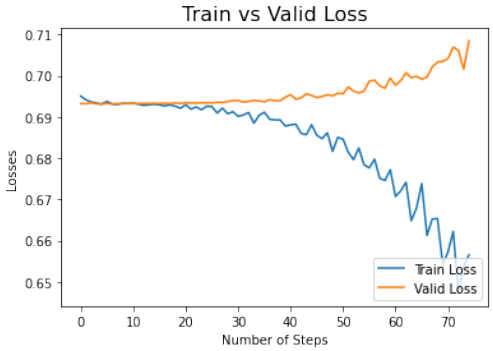
```
1. parameters =
[{'penalty': ['l1'],
 'solver':['liblinear'],
 'C': [0.001,0.01,0.1,1,10]}]
2. parameters =
[{'penalty': ['l2'],
  'solver':['lbfgs','liblinear',
  'sag'],
  'C': [0.001,0.01,0.1,1,10]}]
3. parameters =
[{'penalty': ['elasticnet'],
  'l1\_ratio':[0.25, 0.5, 0.6],
  'solver':['saga'],
      'C': [0.001,0.01,0.1,1,10]}]
```

To total number of hyperparameter combinations searched was 35 amounted to about 2-3 total days of processing (as seen on the **'Time elapsed'** statements underneath all the groups).

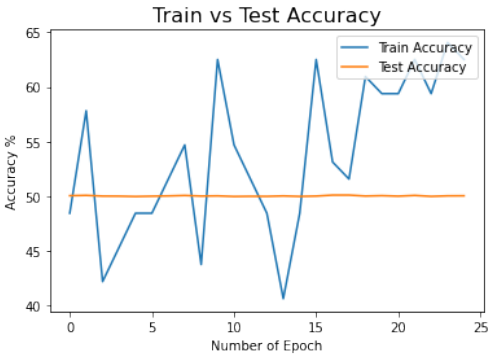
8.2 Neural Networks with Generated Features

8.2.1 Fully connected networks

We started by using fully connected networks to train the board features, and concatenated with the text features from LSTM. As Figure3 shown, the model gets to the arena of overfitting fairly quickly and performs worse on the test and validation set. The situation hold even if we switch to different batch-size[2] or learning rate, the model performs no better than random guess.



(a) trian and validation loss

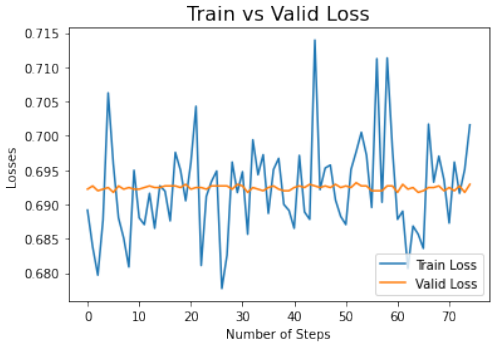


(b) Train and set test accuracy per epoch

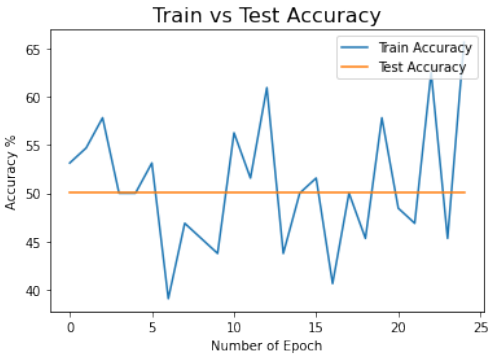
Figure 3: Using fully connected networks with learning rate 0.001.

8.2.2 Convolutional Networks

Given the poor performance outcome, we now apply a slightly more sophisticated model structures where we applied a sequence of convolutional layers to handle the board features. We have explored different combinations of convolutional layer and interacts with varying learning rate $0.01, 0.001, 0.0001$.[3] The model performance did not change much given the learning rate changes. As Figure 4 shown, the model does not converge and we observed training loss jumping around.



(a) trian and validation loss



(b) Train and set test accuracy per epoch

Figure 4: Using convolutional networks with learning rate 0.001.

8.2.3 Discussion

In general, we are surprised to find out that, regardless of the different model structure to apply to the board features, the neural model performance poorly. Only slightly better than random guessing with an accuracy around 51%. We believe that there are two potential reasons that may lead to this:

1. Model architecture is not sophisticated enough. Despite the different structure we have tried, we may not find the optimal model to train the go board data.

[2] We explored batch size 64 and 128, we choose batchsize = 64 to present the results here.

[3] We also explored changing learning rate schedule using $StepLR()$ and $ReduceLROnPlateau()$, however the results remain similar as the one presented here.

The inputs we have is unique. The board is an $(19 * 19)$ matrix with $\{0, 1, -1\}$ as input range. Applying a sequence of convolutiona layers might not be able to fully capture the patterns of the board.

2. Board features and text features are combined using simple concatenation. Essentially we are doing an binary classifications on pair of the board and comments. However, if only through simple feature concatenation, we would loss the effects of interactions of the two features. Hence leads to poor outcomes of the model.

To conclude, the overall performance of the neural model with directly extracted features. The reason behind it is two-fold. First, with current model structures we failed to extract useful board features. Second, concatenation of board and text feature overlooked interactions between the two inputs. With those in mind, in the next section we will explore neural networks using pretrained text and board features, we will also tested different ways of combining the board and text inputs.

8.3 Neural Networks with Pretrained Features

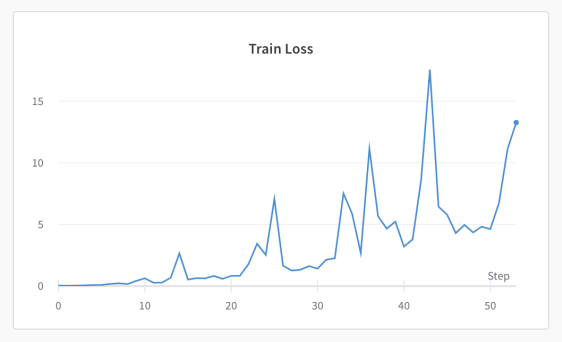
8.3.1 Hyperparameter Search

We experiment with different learning rates on the dot product model: 0.0003, 0.001, and $embedding\_size^{-0.5}$ which is around 0.044. The latter two all diverge as shown in (a) and (b) in Figure 5.

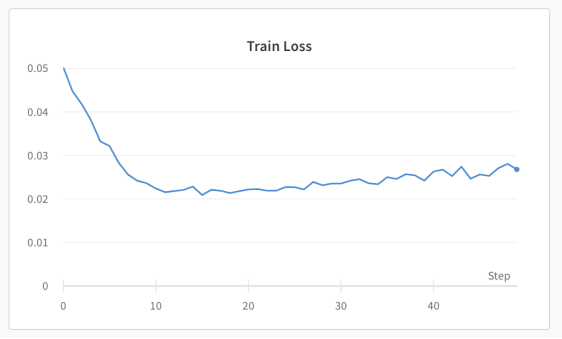
All of our models have warmup of 4000 steps as used in other works like Akoury et al. (2019). To note that Akoury et al. (2019) is in the field of machine translation, which could be very different from our multimodal situation.

8.3.2 Dot product outperforms concatenation

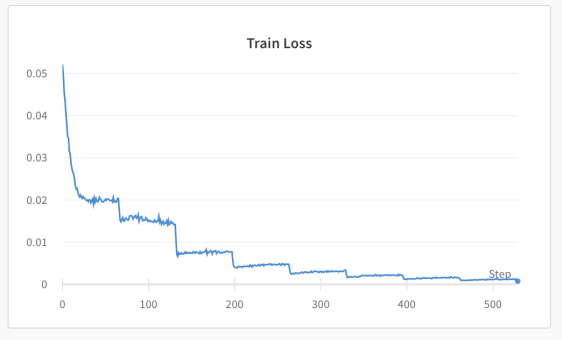
We first did a preliminary comparison between dot product and concatenation to combine the pretrained board and text features. As shown in Figure 6, the concatenation model does not seem to be learning anything, while the dot product model is continuously improving. The concatenation model's training loss and validation accuracy remain the same, while training loss goes down for the dot product model and validation accuracy is also improving. After these two experiments, all our experiments are done using the dot product model.



(a) Learning rate = 0.044



(b) Learning rate = 0.001



(c) Learning rate = 0.0003

Figure 5: Hyperparameter search for learning rate for combining pretrained board and text features. The best learning rate is 0.0003.

8.4 Different batch sizes

We experiment with using batch size of 128 and 512, and both seem to give similar results after same number of epochs, as shown in gray (batch size 512) and brown (batch size 128) lines in Figure 7.

8.4.1 Finetune text embedding

We also suspect that not fixing the Transformer model that is used to extract textual features but finetuning that model as well might help. However, althought the training has not finished yet, the yellow

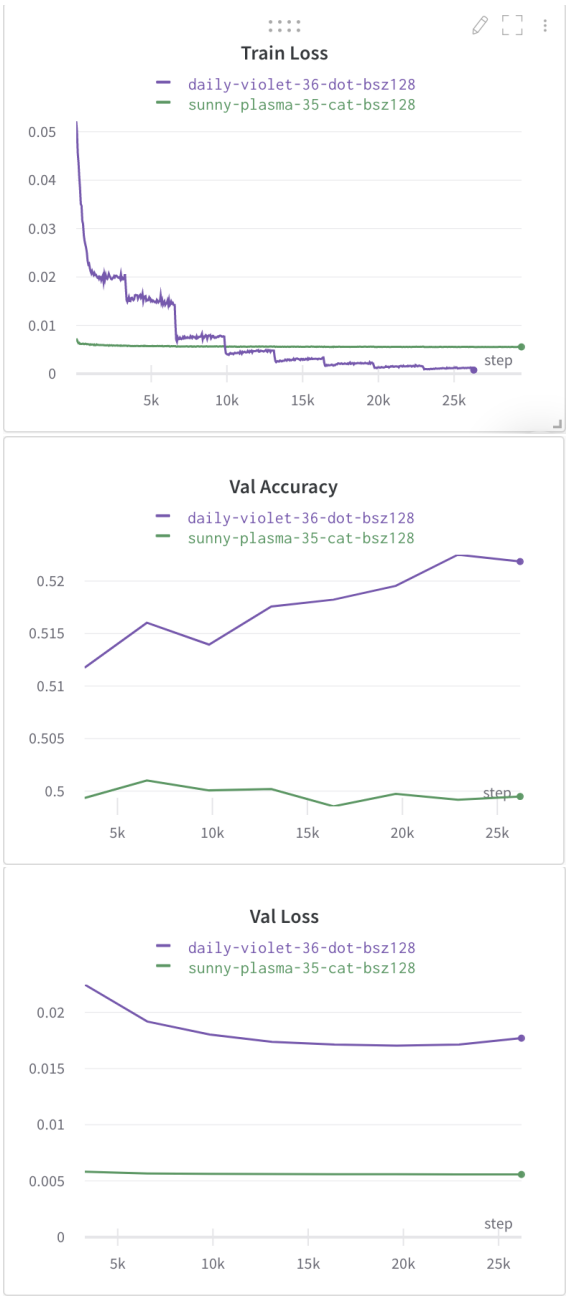


Figure 6: Concatenation model's validation accuracy doesn't change much, but Dot product model's validation accuracy is improving

line that represents the finetuning model in Figure 7 follows the exact same trajectory as the brown line that represents not passing gradients through the textual model.

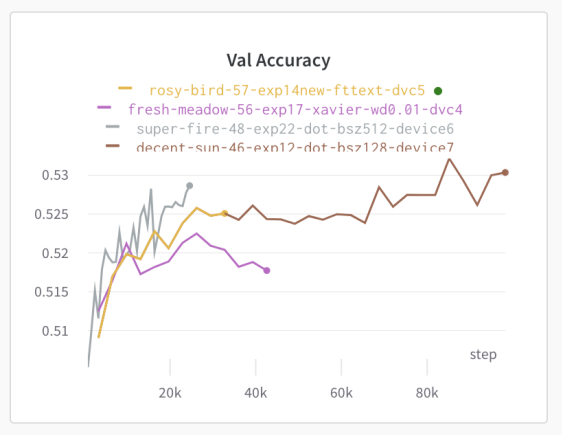


Figure 7: Validation accuracy of dot product pretrain-combine models trained with different batch sizes,

8.5 Use xavier uniform initialization and L2 regularization

Xavier uniform initialization is known to prevent the weights from being initialized too small or too large. Also, as our best model (128 batch size dot product learning rate 0.0003) cannot go further from 53% validation accuracy, we are suspecting overfitting on training set and thus trying to add L2 regularization by setting the weight decay in Adam optimizer to 0.01 as suggested on the Internet. However, as shown by the pink line in Figure 7, validation accuracy goes down after around 40k steps and we thus stop training this variant of our model.

8.5.1 Results of the best model on validation accuracy

As shown by the above experiments, the dot product model trained with batch size 128, learning rate 0.0003, not finetuning text model, xavier uniform initialization or L2 regularization is the winner. The results on test and train using the checkpoint at epoch 26 step 1300 are in Table 4. [4] Using only the last checkpoint is better in recall for test set than averaging over 5 checkpoints. Surprisingly, all the scores of training set is worse than on test set, especially that the recall is only 41.19%. This model has around 53% accuracy on validation.

[4] We tried averaging over the last five checkpoints but the result is worse.

Split	Accuracy	Precision	Recall	F1
Test	50.10	50.09	52.23	51.14
Train	49.81	49.77	41.19	45.08

Table 4: Dot product model of combining pretrained features result (reported in %)

8.5.2 Discussion

In summary for using pretrained features in a neural model, we find that using dot product is more effective than concatenation. This is possibly due to that our task requires multimodal interaction as shown in Hessel and Lee (2020). We would be interested to see if using EMAP to convert our dot product model will make it back to 50% accuracy, but we need to first improve our dot product model to better accuracy to be able to show statistical significance. There are a few things we could improve on.

Explore more complex combination architecture One thing that stopped us from doing more hyperparameter tuning was that the step of inputting board features takes extremely long time and cannot be easily optimized. KataGo uses 22 game specific features like ladder and territory as input to the neural model. Forward pass in the neural model does not take a long time but collecting the features does. At first we just wait for the model to train for days because we are not able to store the hundreds of GBs of pretrained features. Later we thought of storing just the game specific feature inputs which is much more manageable: 6 GB for training set and just do the forward pass during training. This allows us to finish comparing these different models but we don't have more time to compare using ReLU, having more hidden layers, and using multiheaded attention, etc. We will leave that for future work.

Input the pretrained board feature correctly Another thing that could have impacted our result is that we are only using the correct board instead of the whole board history. KataGo's input features ask for the board history, but we only match the current board with text without keeping the history. Thus when we input the features, we input the steps just sequentially from top left to bottom right, without using the actual sequence. After consulting with the author of KataGo, what we should do is to leave the board history dimension as all zeros, and copy the ladder information for those steps. The model is trained without history with some probability so it should be able to handle that, but having the history is better than not having it, and having the last move's history is the most important (which we do keep). This means that our input is different from the input in training, and potentially our board representation is broken. Future work would be to leave those history blank and to also incorporate the real history.

8.5.3 Other directions we tried

Besides the results we showed above, we also tried to build other model or methods to improve the performance. To test the effect of the pretrained model on shallow model, we designed to extracted the features from boards and comments and fed them to the logistic regressioon model. Due to the limitation of memory(we tried 300GB but it still couldn't fit the whole data), we couldn't finish the running. In addition, we also tried to build an end-to-end deep neural network without any pretraining step to test if the end-to-end works better than pretrained model with enough data. However, The model took too long to handle the board features and we didn't managed to finish the training.

9 Conclusion and Future Work

We propose a new task of matching go boards and review comments. Solving this task will help us better understand how AI comes up with suggestions of certain moves. Even though we are facing an extremely hard problem, we still made some progress and learned from the process.

In terms of the baseline, we showed that there are a lot of limitation for the shallow model. Less parameters allows the model to finish training and testing a big amount of data and do the hyperparamter tuning in few days. However, it also makes the model hard to find the complex relationship in the data. With the help of the pretraining features, we expected to see

an improvement compared to the raw data. Moreover, with the help of PCA, the baseline model can get a better 'understanding' of the data and further improve its performance.

For deep models without pretraining, it requires much more time to converge and maybe a more sophisticated model structures. Due to the size of the data, data processing becomes increasingly time consuming. Also, the board input data is sparse and we would like to add domain knowledge to the model, we basically provided a lot of manually engineered features to the model. Beyond that, deep learning model is more flexible so that we have more choices to fit the data. Deep learning model enables more data features to be preserved. In our case, we chose to use transformer to deal with the text data and learn the global features. For board features, CNN shows its ability to extract the local features. However, due to many reason mentioned in the previous section. Neural network model without pretrain does not outperform our baseline by a lot.

Borrowing the idea from unsupervised learning, we also tried to pretrain those two deep models separately without any labels and get the features. Our combination of pretrained features show that multiplicative multimodal interaction might be essential to the model regarding how our task is set up. This can only be evaluated when we have more statistically significant improvements.

We noticed that all results we have gotten for now are not good. One of the reasons may be the weak relationship between the board and comments. However, another more important reason may come from our data. Since we don't have curated dataset, we have to generate the negative samples by randomly sampling. In this way, it's very easy to get a 'reasonable' comment as the negative sample for the board. In this case, it will make the matching even hard for models.

The data might need further processing. The sentences often refer to different positions in the board marked by triangle and squares, or A and B, etc. We did not explicitly represent these to allow matching between markers mentioned in the text and positions in the board. The numerical position like D16 could potentially be learned to map to (4, 16), but where these markers are need to be preserved in the board input. Also in the commentïŒ there exists conversation between student and teacher. Preserving the historical conversation could also help the understanding the relevance of comment.

Besides some unfinished work mentioned above, we also have several good directions to work on as the future work:

1. Better field specific data processing as mentioned above.

2. We randomly sampled the negative examples. Future work could involve filtering out the negative examples that could also work to match with this board, while not being too different to make the task too easy. In addition, we can also introduce sentiment models to classify the positive comments and randomly select the opposite sentiment comment as the negative one to avoid confusion.

3. Use the more correct way of using pretrained board embedding by leaving the history blank if we are not using it, or actually incorporating the history.

4. Test if the multiplicative multimodal interaction is required as pointed out in Hessel and Lee (2020).

5. Allowing the model itself to learn the concatenation may be a better way. Attention mechanism can be one solution to combine two features from text and board and also provide better interpretability.

6. We could record human performance to see what we would expect from models and if there are too many different comments that could be made to the same board position that this task is actually impossible.

Acknowledgments

Dr. Lyle Ungar for his energetic lectures!!

... and to all of our *amazing* TAs:
Tejas Srivastava
Mihir Parmar
Mohit Kumaraian
Siyun Hu
Shaozhe Lyu

Michael Zhou
Hanwen Zhang
Pooja Consul
Qingrong "Margaret" Ji
Gautam Ramesh
Kenneth Shinn
Viraat Singh
Pengrui Wang
Ruiming "Ray" Wu
Yang Yan
Yide Zhao

References

2020 (accessed November 16, 2020). Mosesdecoder. https://github.com/moses-smt/mosesdecoder.

Nader Akoury, Kalpesh Krishna, and Mohit Iyyer. 2019. Syntactically supervised transformers for faster neural machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Jon Arno, Matthias. 2016 (accessed November 16, 2020). The go teaching ladders. https://gtl.xmp.net/.

Zellig S. Harris. 1954. Distributional structure. *WORD*, 10(2-3):146–162.

Jack Hessel and Lillian Lee. 2020. Does my multimodal model learn cross-modal interactions? it's harder to tell than you might think! In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 861–877, Online. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. 2019. Visualbert: A simple and performant baseline for vision and language.

Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. 2019. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Advances in Neural Information Processing Systems*, volume 32, pages 13–23. Curran Associates, Inc.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc.

David J. Wu. 2019. Accelerating self-play learning in go. *CoRR*, abs/1902.10565.

David J. Wu. 2020. Accelerating self-play learning in go.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2020. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, page 1â21.

Bowen Zhang, Hexiang Hu, Vihan Jain, Eugene Ie, and Fei Sha. 2020. Learning to represent image and text with denotation graph. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 823–839, Online. Association for Computational Linguistics.