Elo_Prediction

December 8, 2025

1 Predicting Elo Scores Based on Player Performance through Board States

1.1 Team Members

1.1.1 Alberto Garcia Roberto Palacios Logan Druley

Elo rankings are scores assigned to chess players based on their performance against opponents. Elo scores are calculated by subtracting a player's expected score from their actual score and then multiplying that result by a factor K that controls how fast ratings in a given system are meant to change. The result of this calculation is then added to the player's rating. The formula for this calculation is:

$$R'_A = R_A + K(S_A - E_A)$$

However, a player's Elo rating takes a signficant amount of time to reveal itself. A player must take part in several games before their performance reveals their current skill level. Our plan is to take annotated chess games with stated Elo rankings for each player and train a transformer to read the board states of games. Based on the moves a player takes in relation to their opponent we want the transformer predict the Elo ranking of said player.

[9]:
```
# Import dependencies
import os
import sys

# Add parent directory to path for imports
if os.getcwd() not in sys.path:
    sys.path.insert(0, os.getcwd())

# Import functions from preprocess_data.py
from scripts.preprocess_data import decompress_and_parse_pgn, create_datasets
from src.data.parser import parse_pgn_stream, split_dataset
```

1.2 Data Preprocessing

Here we start by decompressing and then processing 90 Mb worth of annotated games to prepare them for model training. The games are parsed into relevant metadata including the white and black player's Elo rating and also transforms game turns into FEN strings. This transformation allows use to process the entire board's state which in turn gives us the opportunity to use self-attention to discover the importance of each move in the larger context of the game.

[10]:
```
# Process the Lichess PGN file
input_file = "lichess_db_standard_rated_2013-01.pgn.zst"
output_dir = "data/processed"

# Decompress and parse the PGN file
games = decompress_and_parse_pgn(
    input_path=input_file,
    output_dir=output_dir,
    max_games=None,  # Process all games (set to a number like 10000 for␣
 ↪testing)
    batch_size=1000
)

# Create train/val/test splits
train_games, val_games, test_games = create_datasets(
    games=games,
    output_dir=output_dir,
    train_ratio=0.8,
    val_ratio=0.1,
    test_ratio=0.1
)

print(f"\n Processing complete!")
print(f"  Total games: {len(games):,}")
print(f"  Train: {len(train_games):,}")
print(f"  Val: {len(val_games):,}")
print(f"  Test: {len(test_games):,}")
```

```
Decompressing and parsing PGN file: lichess_db_standard_rated_2013-01.pgn.zst


Processing | 100,000 games | 145 games/sec: : 100batch [11:30,  7.56s/batch]

Checkpoint: Saved 100,000 games to data/processed\games_batch_100.pkl

Processing | 121,114 games | 141 games/sec: : 122batch [14:20,  7.05s/batch]


  Total games parsed: 121,114
  Processing time: 861.1 seconds
  Average speed: 141 games/second

Saving all games…
  Saved to data/processed\all_games.pkl

Splitting dataset into train/val/test…

  Train: 96,891 games (80.0%)
  Val:   12,111 games (10.0%)
  Test:  12,112 games (10.0%)
```

```
Saving dataset splits…
  Saved to data/processed\dataset_splits.pkl


  Processing complete!
  Total games: 121,114
  Train: 96,891
  Val: 12,111
  Test: 12,112
```

1.3 Baseline Model: LSTM

Once we've processed the data and then seperated it into train, validation, and test sets, we can then begin to train our initial baseline model. For this purpose, we've chosen an LSTM model as it has the capability to capture relationships across time. However, the performance of an LSTM should be weak in comparison to that of a Transformer with self-attention. As such, this baseline will tell us whether or not we have produce a model with any kind of significant ability to predict Elo.

[8]:
```python
# Train LSTM model with subset of data
from scripts.train_model import train_model, load_dataset
import random

# Check if train_games exists (from previous cell), otherwise load from disk
if 'train_games' not in globals() or train_games is None:
    print("Loading dataset from disk...")
    data_dir = "data/processed"
    train_games, val_games, test_games = load_dataset(data_dir)

    if train_games is None:
        raise ValueError("No data found! Please run the data processing cell␣
 ↪(Cell 4) first.")
else:
    print("Using data from previous cell...")

print(f"\nFull dataset: Train={len(train_games)}, Val={len(val_games)},␣
 ↪Test={len(test_games)}")
print("Creating subset for quick test (5000 train, 500 val, 500 test)...")

random.seed(42)
train_games = random.sample(train_games, min(5000, len(train_games)))
val_games = random.sample(val_games, min(500, len(val_games)))
test_games = random.sample(test_games, min(500, len(test_games)))

print(f"Subset: Train={len(train_games)}, Val={len(val_games)},␣
 ↪Test={len(test_games)}\n")
```

```python
# Train the model with the subset
trainer, history, test_metrics = train_model(
    train_games=train_games,
    val_games=val_games,
    test_games=test_games,
    model="lstm",
    batch_size=32,
    epochs=2,  # Quick test: 2 epochs (~30 min)
    lr=1e-3,
    dropout=0.1,
    embedding_dim=128,
    seed=42,
    output_dir="experiments/quick_test",
    early_stopping=False,
)

# Print results
print("\n" + "=" * 60)
print("QUICK TEST RESULTS")
print("=" * 60)
print(f"Test MAE (normalized): {test_metrics['mae']:.6f}")
print(f"Test RMSE (normalized): {test_metrics['rmse']:.6f}")
if 'mae_elo' in test_metrics:
    print(f"Test MAE (Elo points): {test_metrics['mae_elo']:.2f}")
    print(f"Test RMSE (Elo points): {test_metrics['rmse_elo']:.2f}")
print("=" * 60)
```

```
INFO:scripts.train_model:Train: 5000, Val: 500, Test: 500
INFO:scripts.train_model:Creating dataloaders…
INFO:scripts.train_model:Creating lstm model…
INFO:scripts.train_model:Saved config to
experiments/quick_test\config_20251208_224153.json
INFO:scripts.train_model:Starting training…

Using data from previous cell…

Full dataset: Train=96891, Val=12111, Test=12112
Creating subset for quick test (5000 train, 500 val, 500 test)…
Subset: Train=5000, Val=500, Test=500


Training: 100%|      | 157/157 [09:13<00:00,  3.53s/it, loss=0.0106,
mae=0.0816]
Validating: 100%|      | 16/16 [00:19<00:00,  1.23s/it, loss=0.0101]

Epoch   0 | Train MAE: 0.081552 | Val MAE: 0.081105

Training: 100%|      | 157/157 [08:53<00:00,  3.40s/it, loss=0.0105,
mae=0.0811]
Validating: 100%|      | 16/16 [00:19<00:00,  1.23s/it, loss=0.00991]
```

```
INFO:scripts.train_model:Evaluating on test set…

Epoch   1 | Train MAE: 0.081064 | Val MAE: 0.080164

Validating: 100%|     | 16/16 [00:20<00:00,  1.25s/it, loss=0.0111]
INFO:scripts.train_model:Test MAE: 0.085656
INFO:scripts.train_model:Test MAE (Elo): 171.30
INFO:scripts.train_model:
Training complete!


============================================================
QUICK TEST RESULTS
============================================================
Test MAE (normalized): 0.085656
Test RMSE (normalized): 0.105208
Test MAE (Elo points): 171.30
Test RMSE (Elo points): 210.42
============================================================
```

This initial run only encompasses portion of our data and despite only runing for 2 epoch, we can already see that the LSTM model performs within the 100-250 Elo range that we were looking for as a baseline. The +0.02 gap between RMSE and MAE suggests there no large outliers in this sample of our data but that might change when working with a larger dataset.