sc2model_v1

October 12, 2023

[1]:
```
import torch
from torch.utils.data import IterableDataset, DataLoader
import numpy as np
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import random_split
import random
import file.recordio as recordio
import matplotlib.pyplot as plt

import projects.sc_eval.sc2.sc2_pb2 as sc2_pb2
import projects.sc_eval.sc2.replay as replay
```

[2]:
```
TRAIN_RECORDIO = "/jfs/sc_eval/sc2/train.recordio"
TEST_RECORDIO = "/jfs/sc_eval/sc2/test.recordio"
```

[3]:
```
def ExampleToMinimalTensors(example):
    winner_label = torch.tensor(example.winner_label, dtype=torch.float32)

    p0_race = torch.tensor(example.p0_race, dtype=torch.float32)
    p1_race = torch.tensor(example.p1_race, dtype=torch.float32)

    p0_supply = torch.tensor([example.p0_supply], dtype=torch.float32)
    p1_supply = torch.tensor([example.p1_supply], dtype=torch.float32)

    time = torch.tensor(example.time, dtype=torch.float32)

    features = torch.cat(
        [
            p0_race,
            p1_race,
            time,
            p0_supply,
            p1_supply,
        ]
    )
```

```
    return features, winner_label

def ExampleToAllTensors(example):
    winner_label = torch.tensor(example.winner_label, dtype=torch.float32)

    p0_race = torch.tensor(example.p0_race, dtype=torch.float32)
    p1_race = torch.tensor(example.p1_race, dtype=torch.float32)

    p0_units = torch.tensor(example.p0_units, dtype=torch.float32)
    p1_units = torch.tensor(example.p1_units, dtype=torch.float32)

    p0_upgrades = torch.tensor(example.p0_upgrades, dtype=torch.float32)
    p1_upgrades = torch.tensor(example.p1_upgrades, dtype=torch.float32)

    p0_supply = torch.tensor([example.p0_supply], dtype=torch.float32)
    p1_supply = torch.tensor([example.p1_supply], dtype=torch.float32)

    p0_resources = torch.tensor(example.p0_resources, dtype=torch.float32)
    p1_resources = torch.tensor(example.p1_resources, dtype=torch.float32)

    time = torch.tensor(example.time, dtype=torch.float32)

    p0_misc_stats = torch.tensor(example.p0_misc_stats, dtype=torch.float32)
    p1_misc_stats = torch.tensor(example.p1_misc_stats, dtype=torch.float32)

    features = torch.cat(
        [
            p0_race,
            p1_race,
            time,
            p0_supply,
            p1_supply,
            p0_resources,
            p1_resources,
            p0_units,
            p1_units,
            p0_upgrades,
            p1_upgrades,
#             p0_misc_stats, # TODO: Why does this make % correctly predicted␣
 ↪go down?
#             p1_misc_stats, # TODO: Why does this make % correctly predicted␣
 ↪go down?
        ]
    )

    return features, winner_label
```

```
class Sc2ExampleDataset(IterableDataset):
    def __init__(self, path, transform_fn):
        self.path = path
        self.reader = recordio.RecordReader(path)
        self.transform_fn = transform_fn

    def __iter__(self):
        return self

    def __next__(self):
        message = self.reader.ReadMessage()
        if message is None:
            raise StopIteration()
        return self.transform_fn(sc2_pb2.Sc2ExampleV1.FromString(message))

    def Restart(self):
        self.reader.Close()
        self.reader = recordio.RecordReader(self.path)
```

[4]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)
```

```
Device: cuda
```

[5]:
```
class SimpleClassifier(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(SimpleClassifier, self).__init__()

        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, output_size)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return nn.functional.softmax(x, dim=1)
```

[6]:
```
# TODO: Wrap somehow

def TrainModel(transform_fn):
    ####### Hyperparameters
    batch_size = 512
    learning_rate = 0.001
    epochs = 3

    ####### Get datasets/dataloaders.
    train_dataset = Sc2ExampleDataset(TRAIN_RECORDIO, transform_fn)
```

```
    test_dataset = Sc2ExampleDataset(TEST_RECORDIO, transform_fn)

    # Dataset is pre-shuffled.
    # TODO: Ya but maybe the batches should still be shuffled? I wonder how␣
↪that works with a stream-based dataloader.
    train_loader = DataLoader(train_dataset, batch_size=batch_size,␣
↪shuffle=False)

    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    ####### Define test data runner. TODO: Why put everything in this one giant␣
↪function? Don't be lazy.
    def RunModelOnTestData(prefix="", num_batches=-1):
        num_correct = num_total = 0
        for features, labels in test_loader:
            features, labels = features.to(device), labels.to(device)  # TODO:␣
↪Maybe GPU is excessive/slower here?
            predictions = model(features).squeeze()
            _, indices = predictions.max(dim=1)
            actual_indices = torch.max(labels, 1)[1]
            num_correct += (indices == actual_indices).sum().item()
            num_total += batch_size

            if num_batches != -1:
                num_batches -= 1
            if num_batches == 0:
                break
        print(f"{prefix} Test data got {num_correct} / {num_total} correct, or␣
↪{num_correct/num_total}")

    ####### Init Model.
    ex = next(train_dataset)
    input_size = ex[0].shape[0]
    output_size = ex[1].shape[0]
    # Common advice is that the hidden layer size should be the mean of the␣
↪input + output.
    hidden_size = (input_size + output_size) // 2
    model = SimpleClassifier(input_size, hidden_size, output_size).to(device)
    print(f"Model stats: input={input_size} hidden={hidden_size}␣
↪output={output_size}")

    ####### Train Model.
    # Run a pass through the test dataset before training to confirm there's␣
↪around a 50% chance of getting it right.
    RunModelOnTestData("Test Before Training:", num_batches=100)
```

```
    # Loss Function and Optimizer
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

    # Training Loop
    for epoch in range(epochs):
        model.train()
        total_loss = 0
        i = 0

        # Decrease learning rate for last epoch.
        if epoch == epochs - 1:
            learning_rate *= 0.1

        for features, labels in train_loader:
            i += 1
            features, labels = features.to(device), labels.to(device)
            optimizer.zero_grad()

            # Forward pass.
            outputs = model(features)
            loss = criterion(outputs, torch.max(labels, 1)[1])
            total_loss += loss.item()

            # Backward pass and optimize.
            loss.backward()
            optimizer.step()

#             if i > 100:  # TODO: Change.
#                 break

        RunModelOnTestData(f"After {epoch}th epoch:", num_batches=100)

        train_dataset.Restart()
        test_dataset.Restart()

    RunModelOnTestData("Final run over test:", num_batches=1000)
    return model
```

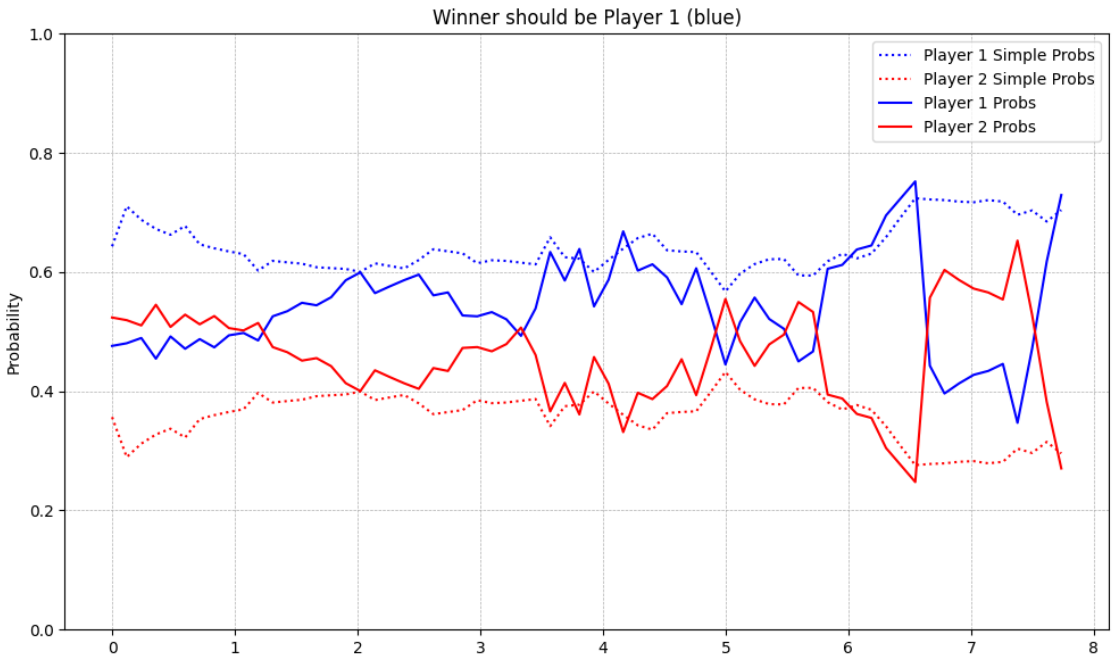
[7]:
```
print("-----------")
print("Training a minimal model with few features.")
model_with_minimal = TrainModel(ExampleToMinimalTensors)

print("-----------")
print("Training a model with a lot of features.")
model_with_everything = TrainModel(ExampleToAllTensors)
```

```
-----------
Training a minimal model with few features.
Model stats: input=12 hidden=7 output=2
Test Before Training: Test data got 23580 / 51200 correct, or 0.460546875
After 0th epoch: Test data got 25373 / 51200 correct, or 0.49556640625
After 1th epoch: Test data got 29024 / 51200 correct, or 0.566875
After 2th epoch: Test data got 29517 / 51200 correct, or 0.57650390625
Final run over test: Test data got 212822 / 376320 correct, or
0.5655346513605443
-----------
Training a model with a lot of features.
Model stats: input=624 hidden=313 output=2
Test Before Training: Test data got 26405 / 51200 correct, or 0.51572265625
After 0th epoch: Test data got 31682 / 51200 correct, or 0.6187890625
After 1th epoch: Test data got 31619 / 51200 correct, or 0.61755859375
After 2th epoch: Test data got 32020 / 51200 correct, or 0.625390625
Final run over test: Test data got 236956 / 376320 correct, or
0.6296662414965987
```

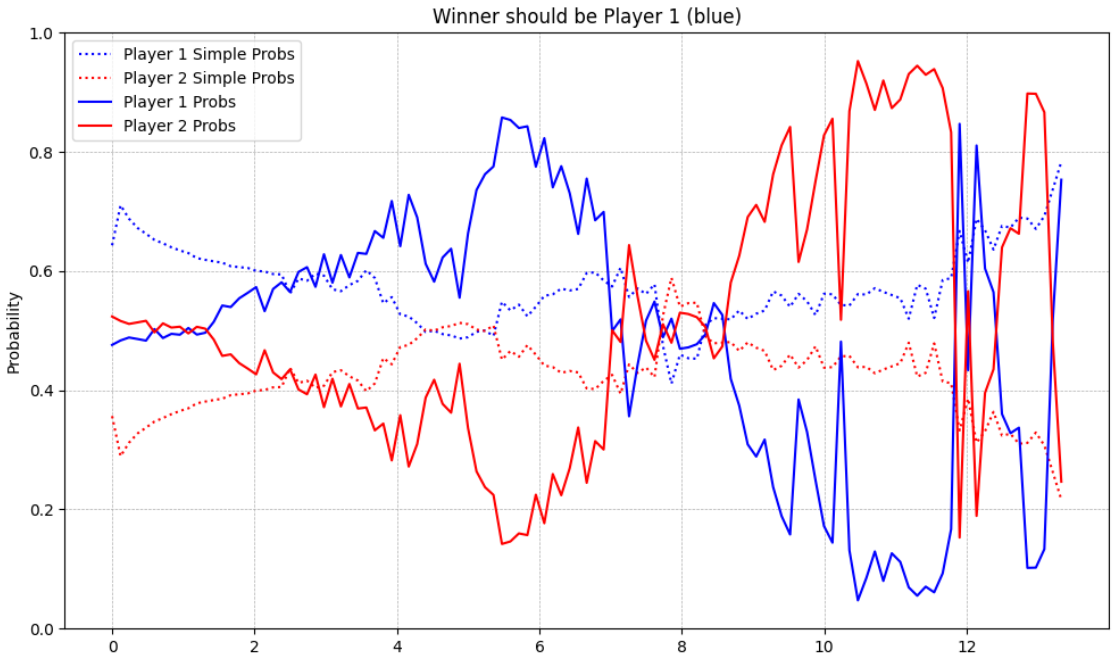
[12]:
```
def GraphGame(path):
    sample_examples = replay.ExtractExamples(path)

    p1_won = True

    p1_simple_probs, p2_simple_probs = [], []
    for features, labels in [ExampleToMinimalTensors(ex) for ex in␣
 ↪sample_examples]:
        p1_won = labels[0].item() == 1.0
        features, labels = features.to(device), labels.to(device)
        outputs = model_with_minimal(features.unsqueeze(0))
        p1_simple_probs.append(outputs.squeeze()[0].item())
        p2_simple_probs.append(outputs.squeeze()[1].item())

    p1_probs, p2_probs = [], []
    for features, labels in [ExampleToAllTensors(ex) for ex in sample_examples]:
        p1_won = labels[0].item() == 1.0
        features, labels = features.to(device), labels.to(device)
        outputs = model_with_everything(features.unsqueeze(0))
        p1_probs.append(outputs.squeeze()[0].item())
        p2_probs.append(outputs.squeeze()[1].item())

    # Each value is approx. every 10 seconds or 1/6 of a minute of game time.
    # * 0.714 is for the faster speed that basically all games are played on.
    x_values = [(i/6)*0.714 for i in range(len(p1_probs))]

    plt.figure(figsize=(10, 6))
```

```
    plt.plot(x_values, p1_simple_probs, label="Player 1 Simple Probs",␣
↪color='blue', linestyle=':')
    plt.plot(x_values, p2_simple_probs, label="Player 2 Simple Probs",␣
↪color='red', linestyle=':')
    plt.plot(x_values, p1_probs, label="Player 1 Probs", color='blue')
    plt.plot(x_values, p2_probs, label="Player 2 Probs", color='red')

    plt.ylabel('Approx game time (minutes)')
    plt.ylabel('Probability')
    plt.ylim(0, 1)
    plt.title(f'Winner should be {"Player 1 (blue)" if p1_won else "Player 2␣
↪(red)"}')
    plt.legend()

    plt.tight_layout()
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    plt.show()
```

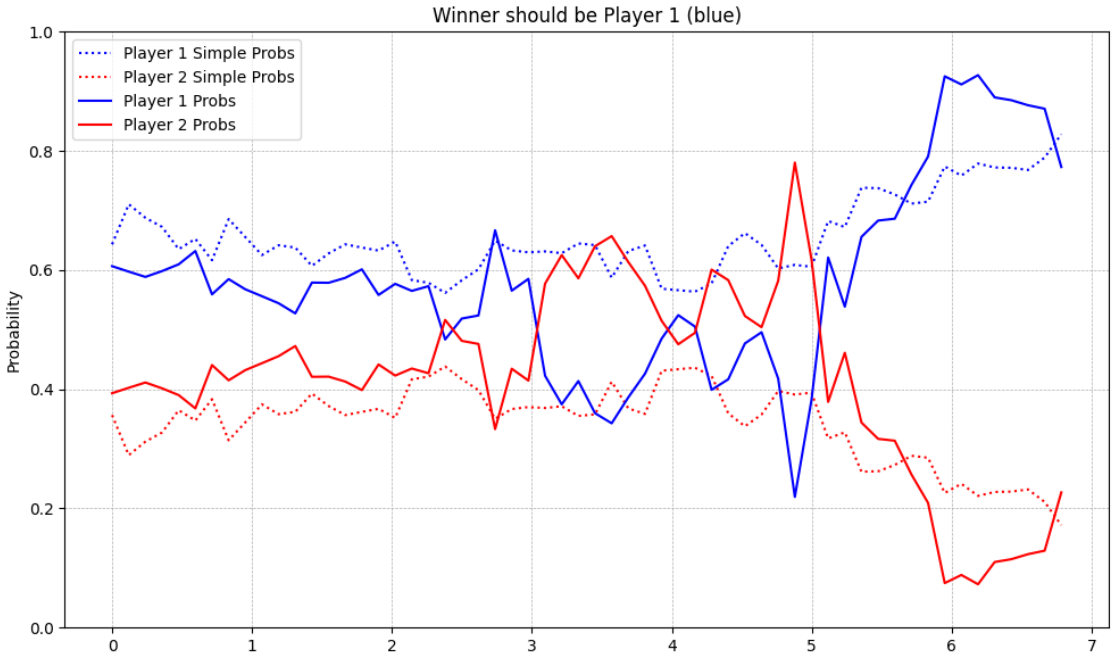
[13]:
```
GraphGame("/jfs/sc_eval/sc2/replays/spawning-tool/replays/
↪fffd6ae3074049abdf0004c8131981e7.SC2Replay")
```



[14]:
```
GraphGame("/jfs/sc_eval/sc2/replays/spawning-tool/replays/
↪ffef4b1528c2824f5683011bca106f19.SC2Replay")
```



[15]:
```
GraphGame("/jfs/sc_eval/sc2/replays/spawning-tool/replays/
  ↪ffe0143ef6ca50dbc29641a4a7955cf4.SC2Replay")
```



0.1 TODO

- Refactor out RunModelOnTestData and make it work with different filters, for example, ignoring time 0-1 minute (and 0-2, 0-3, etc.) to determine how it performs as the game progresses.
- Probably should pre-featurize everything as it makes it harder to do the above?
- Figure out why misc_stats don't actually make the model perform better. (It makes it worse.)
- Look into doing some feature importance?