11/7: revisited data format

daily plan

obviously given the results from 11/2, the nets aren't learning. this is probably a data formatting issue (i thought the net would be able to learn relatively quickly which players were T/CT and judge off of that). it's also theoretically possible that this is a net architecture issue but this is less likely since all the archs i tried weren't able to overcome the "guess the map average" baseline.

i also want to clean up the log file output a bit since it was cumbersome to do that last time.

- [x] add a verbosity parameter to training
- [x] change data format to 4 channels of 5x5 (T, CT, TxCT, CTxT) plus somehow the HP data
- [x] save transforms to disk instead of just train/test splits (for efficiency)
- [x] adapt models to accept the new data format shape
- [x] fix log loss usage
- [x] re-run LR baseline with new format + get log loss numbers for baselines
- [x] implement early stopping via validation set
- [ ] run a few simple experiments to try out the new format

verbosity param

this should be easy enough, just annoying to deal with.

aaand done. i haven't tested but i'm pretty sure it works.

new data format

so when i showed peterx the results from 11/2 he suggested the net probably isn't learning the data format correctly. this kind of makes sense since i sort of just threw the data at the net without any proper formatting or placing any inductive biases. so my first thought here is to enforce some inductive biases on the distance data by separating the Ts from CTs into different channels. this way, the input is shaped like

$$(batchSize, nChannels = 4, numTPlayers = 5, numCTPlayers = 5)$$

i.e. for batch size 3, it would be $(3, 4, 5, 5)$. this still has the same number of elements as before $(10 \cdot 10 = 100 = 4 \cdot 5 \cdot 5)$ but now the data will have more enforced structure since the channels will be (T, CT, TxCT, CTxT) and not just a mishmash that is sorted alphabetically by player name.

i am still unsure what to do inside each of these channels (since there's no "player 1-5" for csgo), so i will probably still sort alphabetically/randomly here. another possible idea is sorting by distance to one of the bombsites but this would be a little trickier i think.

so separating it out into 4 channels and then stacking those is not difficult. the issue right now is separating the 4 channels appropriately into T/CT combinations...

ok i think i got it down. the additional data is being passed as extra channels that are essentially diagonal matrices (2: one for T, one for CT for whether a player is alive). let's test

already getting errors, lol. something related to `.loc` calls, typical. let's see how to fix this.

oh i see. the `.loc` call is asking for specific indices, but the df's index is `(tick, playerid)`. that's not great. how am i going to select this stuff now? do i need to reset the index, select, and then re-do the index? seems silly...

so many tensor operations gone wrong... slowly fixing stuff to have the right shape...

alright, it finally works. had some trouble getting the additional data formatted correctly (was yielding a GIANT diagonal matrix) and with the

somehow i found a game/round that has 10 players per side instead of 5 per side. it looks like the players switched sides somehow? have to tell peterx. for now i'm going to go back and re-split the data and throw out anything with more/less than 5 players per side.

looks like it works now :)

saving transform to disk

this shouldn't be too difficult, just one of those things.

as expected, pretty simple.

adapting models to new data format

this one might be a little tricky since there's a different number of elements per data format. there's also a different number of channels (originally 1, now 6) so the CNNs are gonna be weird too. meanwhile let's run this transform over train/val/test so that i don't have to keep waiting around on it forever.

linear and resnet were actually pretty easy. the convnet is gonna be weird still. mostly because i was originally just relying on different types for the `input_size` param.

ok i think i've got it. now i'm always passing in tuples to `input_size` and resnet/fcnn just get the product of it as their input size. oh and of course they have to reshape the input i guess. but it looks like they already do that. time to test

looks like it runs for FCNN at least. let's see resnet/CNN

and they worked, after a little bit of tinkering. :)

log loss

so the log loss was giving me issues last experiments. it was not following the `BCELoss` from pytorch (bce loss was decreasing, log loss was somehow increasing/becoming inf/nan). looking into this yielded... apparently sklearn's log loss function gives you different results if your array is `float32` vs `float64`, and it doesn't seem to be precision related. unbelievable.

casting to float enforces `float64` so now it works. crazy.

re-run LR baseline, get log loss numbers

this should be pretty quick. log loss numbers is just a different function call, re-running LR is just running the same code with a different transform hopefully.

map baseline

log loss: 16.3849

player count baseline

log loss: 0.5412

LR baseline

acc: 0.7086
AUC: 0.7897
log loss: 0.5183

early stopping via validation set

early stopping implementation is pretty easy, just break if the auc went down. it's probably best to make it so that if the auc goes down 3 (or so) epochs in a row, then it stops. i already implemented for if it goes down it stops; now to make it go down 3 in a row... that means each needs to be increasing. it also means that the actual good model is 3 epochs back.

ok well now i keep a dict of aucs. but i guess this is ok. i still am using the model 3 epochs too late but hopefully it doesn't matter that much. if so, then i have to keep the last 3 epochs of a model on disk, and that gets annoying to do. plus later on i'm probably going to train the entire thing on train+val and then test on test at the end, so i don't think it'll affect things that much.

experiments

these are pretty much just gonna be the same ones as last week. hopefully they work out better this time with the new data format (or at least they give *something* different from each other, lol).

---

summary

things done

- reworked data format
- adapted models
- a few bells and whistles for training: verbosity param, early stopping, saving transforms to disk
- re-ran baseline with new format
- fixed log loss

questions for peterx

- n/a

next steps/remaining action items

- run some experiments with the new data format
- find out what archs/hyperparams work best for this problem