# 10/31: initial net architectures

## daily plan

today the only real objective is to get 3 neural net archs written out. there's a little bit of research involved to make sure that these archs are any good, but i already have 3 in mind: a simple fully-connected net, a CNN, and a ResNet-type arch with skip connections.

- [x] look up win probability models based on neural networks
- [x] write basic "building blocks"
  - [x] resblock
  - [x] linear block
  - [x] cnn block
- [x] write 3 NN architectures
  - [x] FCNN
  - [x] CNN
  - [x] ResNet-style
- [x] implement nice-to-haves: optional activation function, batch norm, dropout
  - [x] different activation functions
  - [x] batch norm
  - [x] dropout
  - [x] different number of layers
- [x] write training/testing code
  - [x] variable model
  - [x] variable batch size
  - [x] variable learning rate
  - [x] some form of cross-validation? or just validation set is enough maybe
  - [x] save model results somewhere
- [ ] start training jobs for the three architectures with a few different options
  - can use prince or local
  - [ ] set up sbatch file/conda env on prince

## research

looking for papers. came across this which seemed promising but it's a really shitty paper and doesn't describe too much. they used a FCNN and some sort of RBF-based NN which honestly just looks like RBF stuff.

also found this medium post which is crappy. they also just used a FCNN with 12 (!) hidden units.

let me go back to my previous notes and see what papers i had put in there. maybe there's something in those papers. i'm just going to skim though and if there's anything interest i'll read further

starting from yifan yang et al used logreg and feature engineering. they do have a NN for "pre-game" modeling which is 2 layers. they tried hidden size [32, 64, 128] and activation [sigmoid, relu, tanh], but they don't have any special architectures.

anders drachen et al they don't really do win probability, nor NNs.

françois rioult et al they don't do WP either.

victoria hodge et al they just use LR and RF.

kuangyan song et al LR and stepwise regression (adding in number of features progressively). only 2 references? nothing on nets.

pobiedina et al only used LR but some interesting social/experience based feature engineering

johansson et al used a lot more ML but still classical ML (RF, gboost, NB, kNN, SVM), no nets.

makarov et al elo is not bayesian at all lol. what are these guys talking about. this also was very much not redrafted by a fluent english speaker (which - is fine, just unexpected that there's broken english in a published paper). also no nets.

so basically back to just searching. this uses a FCC with dropout/batchnorm but it's a medium post. this uses a standard 2 layer NN as well. why do so many papers/projects use 2 layer NNs?

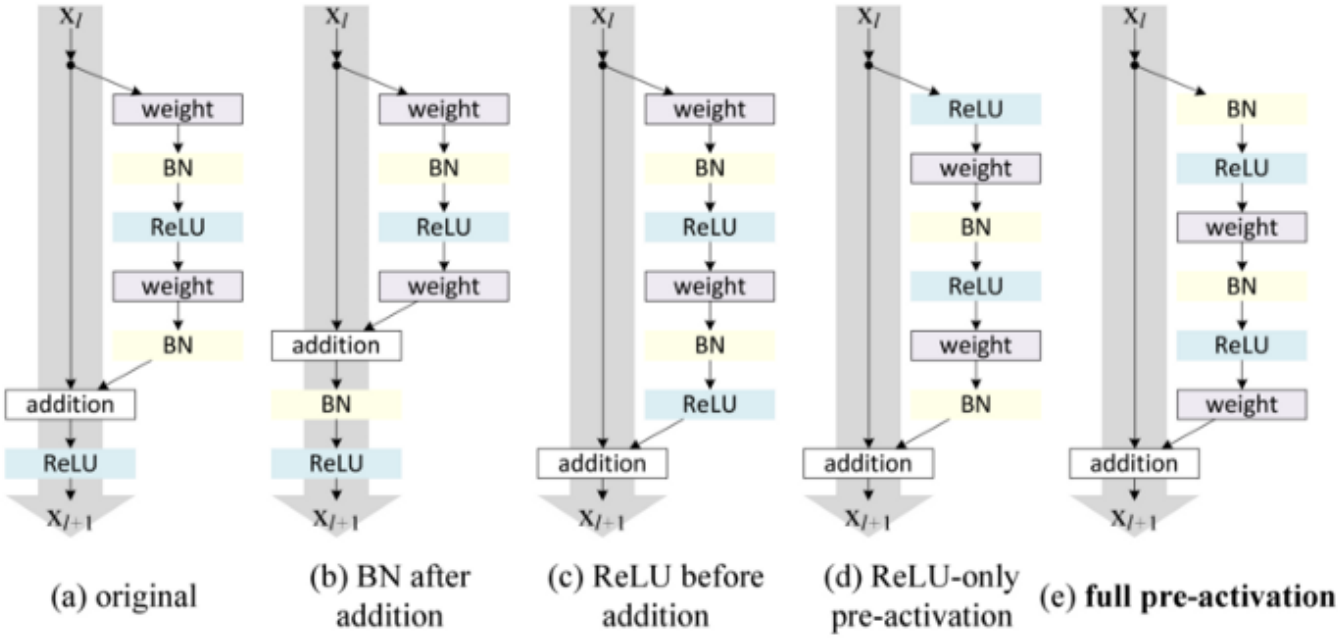
the best i can gather is: nobody has really used NNs for this before. so i am just gonna go with the gut feeling above and make a variable number of layers, with/without batchnorm/dropout, and run a few experiments and see what works, then theorize as to why that worked and iterate.

## NN archs

part of this (as well as CNN) was already kind of written in `model.py`. it has a few of the optional things too but i want to make this a little bit bigger/better/more flexible so that i can just pass command line args to it and have it train a different model.

first, let me write "building blocks": FC layers that have input/output size as attrs and that take in specific activation functions, as well as residual blocks.

residual blocks seem like a good start. found this image




the one that is supposedly best is the bold one but this is [from a medium article] so who knows. i'm gonna go with the one that makes sense to me which is the middle one (c), without the batch norm for now.

linear block also looks easy.

cnn block is a little bit more difficult since there are more params that are possible i guess. but as long as i pass stuff in correctly it should be ok.

ok, now to write a fcnn that can take args for size or something like that.

fcnn was easy, and as expected cnn was a little bit more complicated (with calculating the linear layer size at the end, mainly). now for the resnet

resnet was super easy since it's just a slight variation of linear FCNN anyway. i already have different number of layers implemented (through the kwargs in the initialization of the object) and interchangeable activation functions. at the end of every net there's a softmax.

actually does it matter if i use softmax or sigmoid at the end? interesting thought.

anyway, now implementing optional batch norm and dropout for each of these nets.

batch norm was a little weird, because it somehow needed me to pass in a specific number of features to realize what the batch is. d/m, got it done. now for dropout.

dropout was super easy. now for writing train/test code. this should be ok, just gotta pay attention to how to get the train/val/test data

## train/test code

i already have some training code written in `train.py`. maybe this should be renamed to something else since i also want to include the testing code here. but who cares.

looks like this training code that i already wrote is gonna have to be completely rewritten. but that's ok.

there were a few issues (wrong shebangs, had to use ModuleList instead of regular list)

issue when testing: realized that i can't have two outputs. it's just a single win probability so i don't actually need a softmax at the end and two output neurons, i just need one output neuron and the sigmoid i mentioned earlier.

training code is working, testing code too. my initial tests seem to show that the model isn't learning at all which is pretty bad. i'm running a run through the full dataset now to see if that fixes anything (hopefully it does), otherwise i might have a hard-to-debug problem here. other than that though, i think i'm pretty set for today. all i have to do is run the experiments so that nov 2 i have some results.

bad news. it's not learning at all. lol. now what?

turns out that you can't do an activation right before the sigmoid because... you always end up predicting the positive class. duh. it works now, and it learns, although at least the run i did looked like it was learning bogus. but let's see what happens with other options.

---

## summary

### things done

- training/testing code
- 3 neural net archs: FCNN, CNN, ResNet

### questions for peterx

- n/a

### next steps/remaining action items

- set up sbatch/conda env on prince
- run some experiments so that nov 2 i can look at the results and iterate model architectures