Stephen's Work Log

Stephen Malina

February 25, 2020

Contents

Minutes 1

Those present Stephen Malina

Date January 29, 2020

List of topics

1.1 Kipoi Dropout Research

My current understanding is that, assuming DeepSea is a PyTorch model in Kipoi (will verify shortly), I should be able to run a simple function like the following to activate Dropout in the Kipoi model (source 1, source 2).

```
def apply_dropout(m):
    if type(m) == nn.Dropout:
        m.train()

model.eval()

model.apply(apply_dropout)
```

This means that the next step is to figure out the best way to generate repeated predictions for the same sequence using Kipoi while ensuring that we use the same dropout mask on predictions for wild type and mutated sequences. Unfortunately, this most likely means we won't be able to use kipoi-interpret's out-of-box in-silico mutagenesis code.

Minutes 2

Those present Stephen Malina

Date January 30, 2020

List of topics

2.1 DeepSea Uncertainty Estimates

Huzzah! I'm able to force DeepSEA to apply Dropout at evaluation time using the strategy I mentioned yesterday. That said, one thing that's not clear to me is whether it's OK that not every layer is using Dropout... For now, I'm going to assume it is and flag it as something to follow up on.

Now, the next step is to figure out the best way to ensure that Kipoi uses the same Dropout mask for original and diff predictions.

2.1.1 Verifying that we don't need precision for classification

I found the appendix to Yarin Gal's "Dropout as a Bayesian Approximation" paper and I'm pretty confident that for classification that the predictive variance is just the empirical variance of the predicted probability.

One question that I'm putting aside for now is the question whether 0-normalizing is necessary.

2.1.2 Verifying that torch uses 1 Dropout mask for an entire mini-batch

So it's looking like torch does **not** use the same Dropout mask within the same mini-batch. This means either we:

1. Hot swap the torch dropout layers with custom ones which use 1 mask per mini-batch.

2. Use the standard dropout layers and accept that we'll lose the $\mathrm{Cov}(X, Y)$ term from the

$$\mathrm{Var}(X - Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) - 2 \cdot \mathrm{Cov}(X, Y)$$

   formula for the variance of our predicted difference.

One caveat with option 1 is that, even if torch did use the same mask for the mini-batch, I'm not sure that we'll be able to fit all 4000 mutated sequences in the same batch. I *think* we will but will have to verify.

I think ultimately we should go with the first option, but I'm tempted to start with the second just to get all the scaffolding in place and then we can switch.

2.1.3 Working with Kipoi to get uncertainty estimates

The key question determining whether we can use the built-in mutagenesis code is whether Kipoi makes all the predictions as part of one batch. If it doesn't, that precludes us from ever going with option 2 and calls into question whether it makes sense to use the built-in mutagenesis code in the first place.

Minutes 3

Those present Stephen Malina

Date February 02, 2020

List of topics

3.1 Kipoi Uncertainty Estimates Cont.

3.1.1 Custom Dropout

After fiddling around a bit with the code and reading some helpful PyTorch forum posts, I got custom Dropout working. I'm using a **barely** modified version of SalesForce's `LockedDropout` class code to do Dropout with a constant mask (within a batch). I'm using an also **barely** modified version of this `convert_layers` code to replace the original `Dropout` layers with my own.

3.1.2 In-silico mutagenesis with uncertainty

I now have code that can generate batches of (currently un-)mutated sequences and feed them to DeepSEA. Unfortunately, this is really slow...

3.2 GPU Memory Issues

I'm encountering weird behavior where Kipoi / Torch allocate a huge amount of memory (¿10 GB) just to make predictions for a single 1000-record batch of data. Now, since each sequence is 1000 base pairs, I get that this is a non-trivial amount of data, but there's no way it requires 10 GB just to load and make predictions on. There are two other pieces of evidence that convinced me that something else must be going on:

1. The 10 GB stick around after I make the predictions until I free the model
2. Calling `predict_on_example` doesn't cause the same issues even though

Managed to resolve these by just using small batch sizes.

Minutes 4

Those present Stephen Malina

Date February 04, 2020

List of topics

4.1 Kipoi Uncertainty Estimates Cont.

Managed to get mean, variance, and covariance estimates using all the stuff I've been working on this past week. In the process, I made two graphs which seem worth capturing here.

The first depicts predictive variance versus predictive mean with colors representing different sequences (each dot is for a maybe mutated variant of the sequence). As expected, predictive variance increases as predictions become more uncertain (become closer to 0.5).

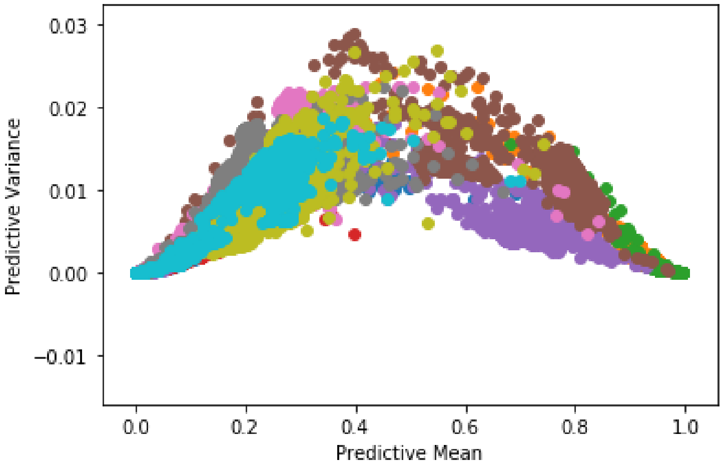

Figure 4.1: Plot of predictive mean vs. predictive variance. Colors code for different sequences.

Minutes 5

Those present Stephen Malina

Date February 06, 2020

List of topics

5.1 Misc.

I spent most of my work time today reading about MR-Egger while waiting for my run to finish. After that, I realized that I'd implicitly been leaving a bunch of un-mutated sequence predictions at the end of the final batch for each sequence in the data. Fixed that by making the batch size an even divisor of 3000. Altogether, not the most productive day...

Minutes 6

Those present Stephen Malina

Date February 10, 2020

List of topics

6.1 Past few days

I forgot to log some of my work over the past few days here but essentially I've just been futzing around with the post-Kipoi data processing a bit and also reading a bunch about MR-Egger regression, which I now mostly understand.

6.2 Today

Now I'm finally ready to actually run the MR-Egger code in R with NN-generated data.

Spent a lot of time getting all the annoying R stuff setup locally. I think I have stub code that's *this* close to allowing me to run MR-Egger on our predictions. However, before I do that, I need to remember to re-run the prediction code with batch size set to 301 instead of 300 so that there are no extra un-mutated sequences in the batches. I also need to remember to generate new random sequences equally sampled from binding / no binding regions **for FOXA2** not CTCF.

Minutes 7

Those present Stephen Malina

Date February 13, 2020

List of topics

7.1 Past few days

Over the past few days, I got the MR-Egger code working and revamped the random sequence selection code to work with FOXA2 data from

Minutes 8

Those present Stephen Malina

Date February 25, 2020

List of topics

8.1 Recap

I've been bad about recording progress so I'm using this to recap all that's happened since my last entry.

After getting the MR Egger code up & running, we realized that the uncertainty estimates for individual predictions were 'too' high, leading to I^2 scores of 0. This roughly means that Egger's estimates will suffer from 'weak instrument bias'.

Given that, I spent some time looking at Gaussian processes to try and figure out whether there was a principled way to set the dropout rate p that also might reduce or at least explain our large standard errors. I found that the dropout probability emerges in the GP VI approximation derivation as a mixing weight for a two-component mixture of Gaussians in which one component represents the NN weights and one represents 'zero'.

The GP formalism turned out to be not that helpful for figuring out a principled way to set the dropout probabilities. That said, in the process, we did discover a different paper which talks about the Expected Calibration Error (ECE) metric for testing calibration. The computable version of the ECE formula requires first breaking the range $[0, 1]$ into k equal-sized bins. For each bin, we compute the absolute difference between mean predicted probability of samples and true fraction of samples with label 1. And then the ECE is a weighted average (by number of samples per bin) of the bin-level differences.

They use this method to compare different forms of MC dropout and NN ensembling and find that something called *dropChannel* works best.

8.2 Today

I'm working on implementing *dropChannel* and will test it as a substitute for the current dropout layers to see whether the s.e. estimates seem smaller.

Before switching to *dropChannel*, I'm going to try changing all the dropout probabilities in the network to be equal (.5 → .2 after the last conv layer) and see whether that meaningfully changes the networks' predictions. I didn't check super rigorously but it seemed like the s.e.s look similar for the two sequences I sampled with all dropout rates set to .2.

Using *dropChannel* seemingly has not decreased the standard errors and has in fact increased them.