Stephen's Work Log

Stephen Malina

February 2, 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 $\text{Cov}(X, Y)$ term from the

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

   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.

3.2 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 $\texttt{predict}_o n_e xampledoesn'tcausethesameissueseventhough$