1 Practical Considerations Concerning Tensorflow

Most of the computations for my research rely on TensorFlow version 2.3.1, a Python library for machine learning specializing in building models with differentiable, parameterizable composite functions and learning model parameters using gradient descent or other gradient-based optimization methods. TensorFlow is a common platform for researchers and developers working on artificial neural netwokrs, and there are many tutorials and exampes freely available online, so I will not replicate that work here. This chapter assumes the reader already has some familiarity with TensorFlow and Keras (a high-level library inside TensorFlow). The goal of this chaper is to provide the reader with the tools and workarounds to be able to replicate my work without resorting to hacking things together with gradient tape and/or TensorFlow-1-style code.

1.1 Why Not Use Gradient Tape and TensorFlow-1-Stye Code?

Keras offers a high-level environment. Code written in Keras's framework is easier to integrate with other work. Gradient tape is great for hacking something together or debugging, but promotes styles of coding that are less readable and harder to splice together.

1.2 Shared Weights Between Layers

Trainable TensorFlow variables declared outside of any Keras layer will not be automatically added to a Keras model's list of trainable variables. In most cases, this limitation is not a problem; it is intuitive to declare a layer's weights inside that layer. However, sometimes the same variable is needed in multiple distinct layers. To be include a variable in the model's trainable variables, it is sufficient to declare the variable in one layer and pass the variable (or the layer it was initialized in) as an input argument to the __init__ function of the other layers that share that variable. This will work even if the Keras model does not use the layer that declared the variable. [1]

1.3 A Note on Custom Gradients

TensorFlow offers a well-documented means of replacing TensorFlow's gradient computations of an operation with specified custom gradient computations. However, if the operation involves multiple tensors that are inputs or trainable

[1] One could instead declare the variable outside any layers, pass it into the __init__ functions of all the variables that depend on it, and then manually add the variable to the model's list of trainable variables, but I do not recommend this approach. The resulting code will be less readable and much less maintainable.

variables, the standard approach replaces all the gradients with custom gradients. If TensorFlow's gradient computations are sufficient for some tensors but not others, a workaround is necessary.

1.4 Updating TensorFlow Variables After Applying Gradients

To update TensorFlow Variables after applying gradients, it is necessary to track which variables are affected and what their corresponding update functions are. To accomplish this, I store the update functions in a Python dictionary using variable names as the dictionary keys. This dictionary needs to be widely accessible so that layers can add update functions when they are initialized; a simple way to do this is to make the update function Python dictionary a class attribute. The keys need to be unique, but TensorFlow variable names can conflict. It is easy to avoid this problem by checking for conflicts before adding a new update function.

In the standard Keras training paradigm, models are trained using the fit function, a method in the Keras model object. The fit function calls the function train_step, where gradients are applied. To update TensorFlow Variables after gradients are applied, train_step is the function to modify. The only change that needs to be made is adding a function call to all update functions that correspond to the model's list of trainable variables.

Changes to Tensorflow variables in the update function must use the assign command (or its variants: assign_add, assign_sub, ect). Otherwise, TensorFlow will detect your computations lie outside of its computational graph and throw an error.

1.5 Other Considerations

The TensorFlow Probability version 0.11.1 is an extension of TensorFlow mosly used for probabilistic models. The library contains a Cholesky update function, but the function does not properly handle complex inputs. To compute Cholesky updates for complex inputs, users should either write their own implementation or use my code (included in supplementary material).