summary

Krishna Sankar, Billy Vreeland & Tej Azad

2016-03-12

Experimental Package

The deepLearnR package was developed as a class project for Stanford Stats290-Paradigms for Computing with Data under Professor Balasubramanian Narasimhan.

This is an initial version of the package. It works, but has opportunities for optimizations. We have turned on verbose 1:trainer and prints the progress. Later, some of the veboase options will be silent by default, with switches to trun them on, as needed.

Goals

There are at least two sets of users of TensorFlow - first the Data Scientists/Statisticians (who want to implement the machine learning abstractions) and the other AI researchers (who want to create complex computational graphs as part of various experiments). With deepLearnR, we address the goals of the first audience (i.e. to implement machine learning algorithms) leveraging the deep learning paradigms and the scalability of TensorFlow. And this project focuses on a subset of the audience - data scientists who program in R.

The main use case, that we envision, would be developing deep learning models for predictive analysis in R, beyond the narrow use case of images. At first, we see this package being used as part of R ensembles - deep learning models being one or more of the models in ensembles. The data will be in R, the model will be in TensorFlow (leveraging the multi-core and multi machine (future) capabilities of the TensorFlow framework) and the results will be in the R layer. The ensemble models have a lot of advantages with different paradigms and deep learning is an effective toolset to have in an ensemble.

We envision the users to use R for optimizing hyperparameters for various Deep Learning Models - different network architectures, the learning rate, minibatch size, type of networks like rnn,lstm and dnn. While, in the context of this project we are exposing only a few useful (and obvious) hyperparameters that pertain to the network architecture, we are sure that there will be more hyperparameters as the package matures.

Ultimately the benefit is for the R community to leverage the TensorFlow framework as a backend engine, seamlessly across CPUs and GPUs, distributed across multiple machines.

Stacks

There are multiple deep learning stacks that we can leverage - Theano, Torch and TensorFlow. We chose TensorFlow as the initial underlying framework but deepLearnR can be enhanced with interfaces to Theano and probably Torch.

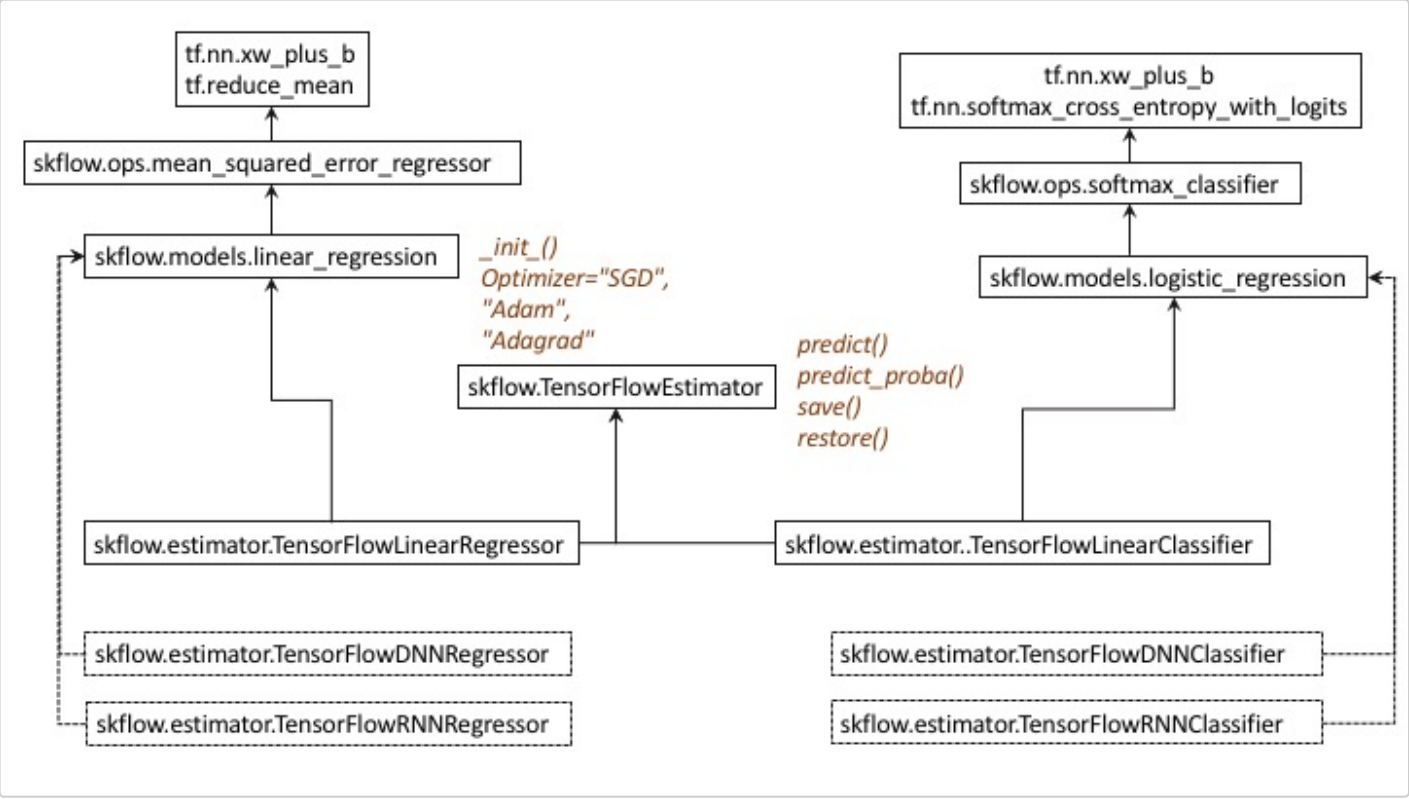
```
Intereatingly, even with TensorFlow there are multiple paths. Currently, as of v 0.7.0, TensorFlow is
divided between python and C++. The graphs are created in python and executed in the C++ layer. So we
use rPython and interface with the python layer. As we were researching, other alternatives like Keras
and Scikit Flow (Skflow) emerged that have interesting abstractions over TensorFlow. Of these, Skflow
is much more streamlined for data scientists' work and its goal matches with our goal i.e. provide ML
abstractions for data scientists.
```

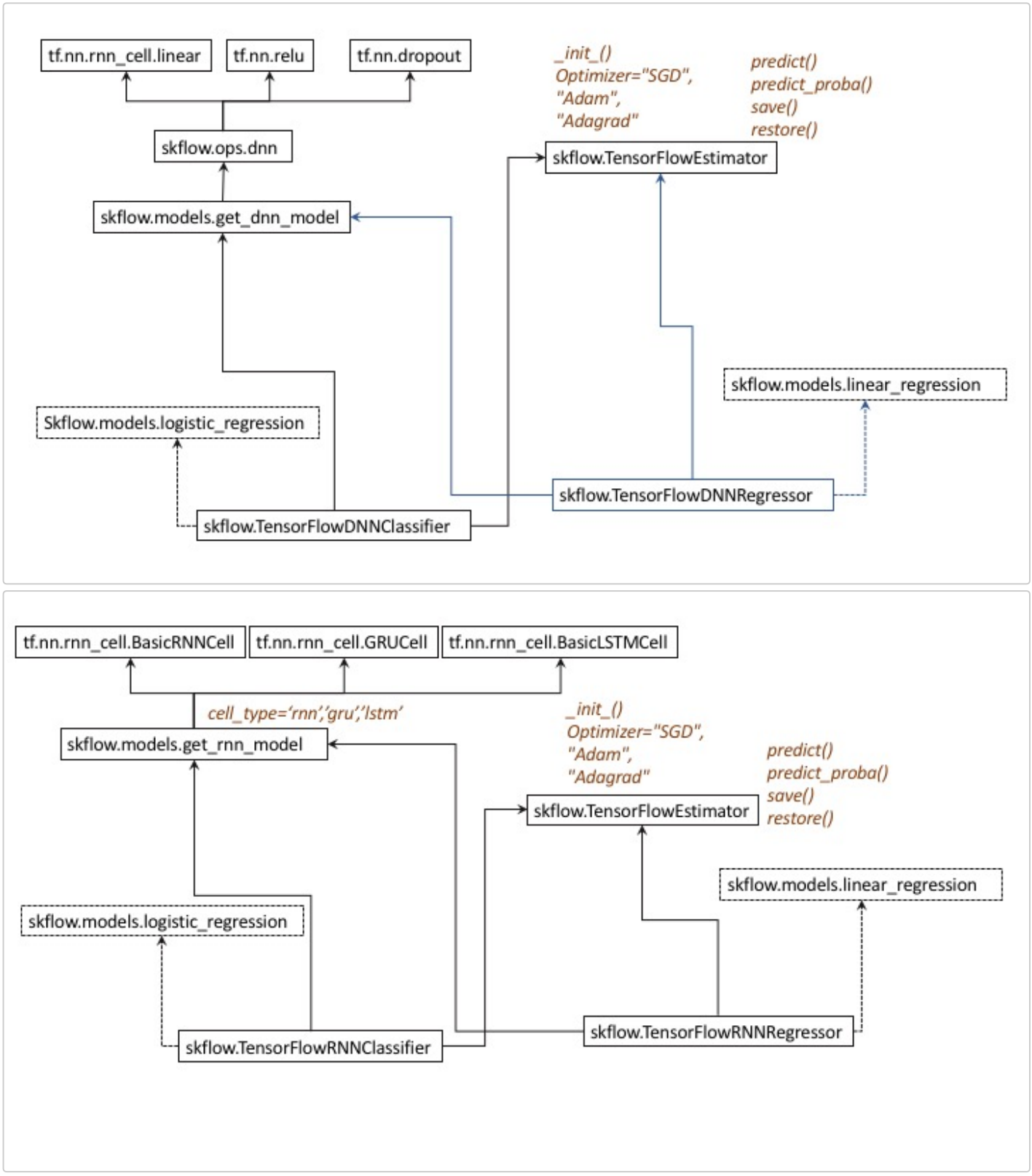
While the focus of development has been on interfacing with TensorFlow via Skflow, we have also included a simple linear regression model that bypasses the Skflow layer and works with TensorFlow directly. This approach does not rely on rPython, but instead generates a python script executed via a system call and utilizes JSON serialization to marshal data and results. At the moment, the Skflow approach is vastly more robust. However, we feel that components of this alternative framework may prove useful as development progresses as it allows for complete access to the TensorFlow API. Given how our understaning of TensorFlow has improved over the course of the project, implementation of a relatively vanilla neural network via the alternative approach is a reasonable next step. In addition, the JSON serialization aspect may prove to be a more effective means of moving larger data sets used in conjunction with the Skflow approach.

While users can write rPython to evoke TensorFlow or Scikit Flow, they will have to stitch together multiple classes, method and abstractions to perform normal ML activities. The goal of deepLearnR is provide a set of unified R interfaces (“structured machinery” as it is called) irrespective of the underlying frameworks. In fact a unifying interface and the capability to select the appropriate framework would be a goal when the package is fully developed.

Interfaces

DeepLearnR exposes interface to most useful abstractions in TensorFlow via the skflow API. First let us start with the class diagrams of skflow as shown below. The class diagrams are instructive and enables us to understand how the various abstractions finally reach the actual TensorFlow graph objects.





DeepLearnR interacts with Skflow at the estimator level, controlling the model via the various parameters. As of the initial version, that level proved to be sufficient. As the sophistication of deepLearnR and the span of problems that it can solve increases, we would need to interact with other abstractions at Skflow, and may be even create a few for interfacing with R.

The interface is deceptively simple, in line with how R interfaces with models viz one function for creating a model, with the needed parameters and another for prediction using that model. It is not efficient or necessary to marshall a model across the python R boundary - so model tags are used instead of actual model objects.

We have deliberately chose to implement a subset of the parameters to keep the work managable w.r.t this

class project.

API Signature

In this section we describe the design in terms of parameters that we have implemented and the features that are acndidates for future enhancements. The actual API themselves are described in the source code as roxygen markup. That way the API documentation always remain current, especially during the initial development.

The create API parameters include modelTag, nClasses, miniBatchSize, steps, optimizer, learnungRate, hiddenUnits, nnType, netType & cellType.

- modelTag - There is no easy way to marshall and unmarshall models from R across to Python and then recreate in a tensorFlow session. So we resorted to the tag scheme where we tag a model and use the tags to invoke them. The tagged models are stored in a list in the Python side.
- nClasses - Number of classes for classifier
- miniBatchSize
- steps - epochs
- optimizer - TensorFlow supports “SGD”, “Adam” & “Adagrad”. But the initial implementation ignores this parameter and defaults to “SGD”
- learningRate - the learning rate of the optimizer
- hiddenUnits - ARchitecture
- The interface can support Linera, Deep Neural Nets and Recurrent Networks with rectified Linear Units and tanh.

Furture Enhancements

The interface can be enhanced with the following parameters, at a minimum. We include this section as a guidance for future work by us and others. We did have a few ideas how to implement these features, but didn’t have the time to implement them. Moreover, thes capabilities come after a robust basic interface and functions are developed, which is what we aim at the first cut with the current version.

- Ability to save and load models via save(modelTag, fileName) and load(modelTag, fileName) functions
- Weight initialization - There are standard methods like Gaussian (with mean and SD as parameters) and range with -sqrt(1/n) to +sqrt(1/n). Other more exotic schemes need a little thought; as we cannot marshall an R function across, we will need named mechanisms 9with appropriate python functions) with parameters for each scheme.
- Control for mechanics like early stopping, dropout, variable learning rate, optimizer like rmsprop and Momentum incluidng Nestrov’s momentum.
- Some of the controls would need appropriate implementations in TensorFlow.

Agonies & Ecstasies

1. Our challenges were getting the various layers viz. R, rPython, python interpreter, TensorFlow python and TensorFlow C++ layer
    - For example rPython will not work with anaconda distributions of python
    - In Mac, installing TensorFlow 0.6.0 on default Python raised errors - because some of the components are secured against changes.
    - Importing TensorFlow via rPython consistently gave the error “cannot import name pywrap_tensorflow” for all of us. We tried various things and finally it disappeared.
    - There were other similar initial errors which disappeared after sometime, still a mystery to us how they were solved. TensorFlow also had a new release 0.7.0 since then.
2. We were able to corral the interfaces to 3 Network Types (“linear”, “rnn”, “dnn”) X 2 Activations(“ReLU”,“tanh”) X 3 rnn types (“rnn”,“gru”,“lstm”) X 3 Optimizers (“SGD”, “Adam”, “Adagrad”).

While this simplifies the interface, all permutations are not valid. There is an opportunity to streamline the interface - may be split into linearNN, DNN & RNN.

3. Once the interface is streamlined, the package offers an elegant flexible way to create different deep network architectures over a distributed TensorFlow framework. For example a 3 layer deep neural network can be created by nnType="dnn" and hiddenUnits=c(10,20,10). Later it can be changed to a 4 layer hiden layer by changing hiddenUnits=c(10,20,20,10). One can work via packages like caret and search for hyper parameters at the R layer while leveraging the scalability of the TensorFlow framework.
4. We wanted the interface to be as complete as possible, and that is why we have included rnn. Basic rnn/lstm works with the current interface, but the interface might have to change to add more complex lstm layers, newer gru networks and LSTM architectures with peepholes.
5. We haven't implemented ConvNets. CovNets have more complex architectures and probably would need richer parameters (than the current interface) to be flexible in defining the architectures incl stride specifications, multiple templates, maxpooling layers et al. So our recommendation is to have a different function to model and predict using Convolution Neural Networks.
6. We didn't get time to test the Optimizers Adam & Adagrad. We used SGD. This is a straightforward addition, but needs to be tested and verified against 2 or 3 datsets.
7. We have implemented two ways of defining and working with TensorFlow - one via rPython and another via system calls. Interestingly the system call might a flexible way to leverage packages like CNTK See future below) where the NDL can be defined and CNTK activated via a system call. We haven't tried this, but the opportunity & the flexibility is there as part of our interface.

Future

Deep learning frameworks are being developed from multiple companies.

Microsoft has released the Computational Network Toolkit a.k.a CNTK. CNTK is similar to TensorFlow in the sense that it is also based on executing a flow graph architecture. CNTK has a Network Definition Language (NDL) and it would be interesting to see if we can take the same interface and implement networks in CNTK. We haven't looked into the feasibility and it depends on the binding exposed by CNTK.

Yahoo has CaffeeOnSpark which has a different take on deep learning based on Hadoop and Spark frameworks. Depending on the bindings, this could be leveraged using SparkR underneath, an opportunity for future projects.