GSoC 2018 PROPOSAL (The Beam Community)

Name and Contact Information -
Name : Anshuman Chhabra
Email : anshuman.lalakers@gmail.com, anshumanc.1996@gmail.com
IRC nick : anshuman23
Github handle : github.com/anshuman23
Phone : +91 9968308825

Title -
TensorflEx: Tensorflow bindings for the Elixir programming language

I. INTRODUCTION AND MOTIVATION:

Recently, with the advent of programmatic and practical machine learning, programmers have been able to integrate applications for the web and the mobile with machine learning and artificial intelligence with great success. This trend has largely been possible because of major organizations and software companies releasing their machine learning frameworks to the public-- such as Tensorflow (Google), MXnet (Amazon) and PyTorch (Facebook). Python has been the de facto choice as the programming language for these frameworks because of it's versatility and ease-of-use.

In my opinion, Elixir is the functional programming language equivalent of Python and Ruby, in that it combines the versatility and ease-of-use that Python and Ruby boast of, with functional programming paradigms and the Erlang VM's fault tolerance and robustness. However, despite these obvious advantages, that pave the way for Elixir as a language of the future (for example, the Phoenix framework for web apps) there is no support for machine learning, yet.

Bearing this thought in mind, I would like to work with the Elixir mentor(s) this summer, to build TensorflEx, an Elixir framework with a Tensorflow C API [1] backend. In the subsequent sections of this proposal, I build upon how I envision the framework to look like in terms of its functionalities, and give cases of languages and frameworks that use Tensorflow as a backend to support my design choices.

II. DELIVERABLES:

At a high level, the proposed deliverables will be as follows:

1. A framework largely resembling the Python machine learning framework Keras [2], which emphasizes ease-of-programming over terse and convoluted code.

2. The framework will consist of Inference capabilities, in that it will allow users to run their pre-trained Tensorflow models (trained using Python for example) and make predictions from their input data using Elixir. Inference is only supported so far because the Tensorflow C API only supports that at the moment. For the Tensorflow framework for Golang too, only Inference is supported [3].

3. Writing tests for each of the functions created.

4. A Blog post detailing weekly progress on my personal blog: http://anshumanc.ml

III. PROPOSED APPROACH:

First off, there are two ways most programming languages can use the Tensorflow C API to create a framework:

1. Create a wrapper around the API, that is, for each of the functions in the API have the same functions in the target programming language. However, this alone is not desirable as such, because the C API is a low-level API. Programming using the C API (and the similarly created target language wrapper) would lead to convoluted and unnecessarily verbose code. It would make more sense to have functions that can use the C API code to provide a higher level abstraction in the target programming language. This would make it easier for users to rapidly utilize machine learning in their applications without worrying too much about how everything works under the hood. Having high level functions would especially make sense for Elixir, where a majority of programmers just want to incorporate machine learning into their applications with minimal effort.

2. Use the C API to provide high-level functions in the target language that can accomplish common tasks required by users. These would essentially include being able to run a pre-trained model by loading the graph and then eventually generating predictions for an input dataset by running a session.

I will combine both the aforementioned approaches for TensorflEx. Functions that are low-level will give experienced machine learning practitioners the flexibility to play around with the code as they wish. High-level functions will essentially emulate the way Keras works by promoting ease-of-use. Moreover, the first priority would be to write the high-level functions instead of porting every C API function to Elixir.

As a case study, I will very briefly discuss how the .NET [4] and Swift [5] Tensorflow bindings have been written. Then I will discuss my approach and the functionalities I will incorporate for TensorflEx. It is important to note that both the .NET and the Swift bindings have been written by users and not by Google. It is worthwhile to discuss only user created bindings as the ones by Google, such as for Golang or Java, have been

written from scratch, have codebases of more than 15000 lines and are virtually impossible to reproduce. Moreover, Google itself suggests that foreign language bindings should be written using the C API. Google also has some guidelines present here [6] regarding what functionalities should be available in a Tensorflow framework, and I will discuss how my proposed approach covers all these as well.

- **.NET BINDINGS:**
  The .NET Bindings are based on the second approach of creating high level abstractions using the C API. While the Github repository does not house any documentation, the way the bindings have been constructed are quite modular. From the over 1400 lines of code that make up the C API header file *c_api.h*, these bindings divide the code into multiple modules. Different functionalities are contained within each module. These are *Buffer.cs, Queue.cs, Tensor.cs, Tensorflow.cs, Variable.cs* among others. As a brief overview, we can take some example use cases. *Tensor.cs* contains functions to create different tensor types; such as String constant tensors which are created from C# byte buffers using the CreateString method [7]. It can also create tensors with custom tensor by specifying a data type, the size and the number of elements using the TFTensor method [8]. There are many other functionalities provided in this file that can be observed by looking at the source code. Moving on, *Tensorflow.cs* contains code that provides functionalities such as setting a status code for a Tensorflow status using the SetStatusCode function [9], getting the dimensions/shape of a tensor using GetTensorShape [10], initialize a new Tensorflow graph using TFGraph [11], etc. The rest of the modules also contain similar code.

- **SWIFT BINDINGS:**
  The Swift bindings are also based on the second approach of high level abstractions. However, here another API in C, based on the original C API, has been written by the repository owner. This second simpler API is used to write the bindings for the target language [12]. The Swift bindings too are similar to the .NET bindings and have been written as a collection of a number of relevant modules.

For TensorflEx, the first aim is to allow users to take a previously trained model and be able to run it in Elixir to generate predictions. Therefore to walk through the proposed high-level capabilities of TensorflEx, I will give an example of getting predictions from a pre-trained model stored as a graph definition. Along with the Elixir functions for doing so, the C (API) backend code that will achieve this is described as well. It is important to understand that while this C code is correct, it does not contain any NIF code.

I will describe how a high-level program in Elixir would work and then list out some of the the low-level functions that will directly be ported to Elixir from the C API.

-> **HIGH LEVEL OVERVIEW OF TENSORFLEX:**

- Assuming the user has a pre-trained graph definition called *graphdef.pb,* we need to have an Elixir function that takes in this file and loads it into a newly created Tensorflow graph.

  ELIXIR FUNCTION
  load_graph(*graphdef.pb*)

  C BACKEND CODE
  ```
  TF_Buffer* read_file(const char* file);
  TF_Buffer* graph_def = read_file("graphdef.pb");
  TF_Graph* graph = TF_NewGraph();
  TF_Status* status = TF_NewStatus();
  TF_ImportGraphDefOptions* graph_opts = TF_NewImportGraphDefOptions();
  TF_GraphImportGraphDef(graph, graph_def, graph_opts, status);
  ```

  The internally created *graph* now has the graph definition from *graphdef.pb* loaded into it. Hence, *graph* can be returned by the load_graph function so that the user can use it for the rest of the program.

- Next, the user would want to get the first input operation in the defined graph. As an example, taking the input operation name to be *“input”* in the graph we will obtain the input operation. This should then be stored in a vector or array and be returned back to the Elixir function. For C, glib GArrays [13] could be used as dynamically growing arrays might be necessary.

  ELIXIR FUNCTION
  get_input_op(graph, “input”)

  C BACKEND CODE
  ```
  TF_Operation* i_op = TF_GraphOperationByName(graph, "input");
  TF_Output input_op = {i_op, 0};
  input_ops = g_array_new (FALSE, FALSE, sizeof (TF_Output));
  g_array_append_val (input_ops, input_op);
  ```

- Similarly there will be a function to get the output operation. For example, the output operation could be called *“output”*.

  ELIXIR FUNCTION
  get_output_op(graph, “output”)

C BACKEND CODE

```
TF_Operation* o_op = TF_GraphOperationByName(graph, "output");
TF_Output output_op = {o_op, 0};
output_ops = g_array_new (FALSE, FALSE, sizeof (TF_Output));
g_array_append_val (output_ops, output_op);
```

- Next, it is important to talk about the input tensors and the output tensors for this example. The user would need to pass the dimensions of the input tensors as *input_dims*, the values of the tensor (*values*) as well as the size of the tensor to be allocated (*input_bytes*). Appropriate resources would have to be allocated for the GArrays so that they can be returned back to Elixir and used later. Moreover, the *input_dims* would be an array representing the dimensions of the inputs. In C, this would be an *int64_t* array. However, in Elixir these values will be passed as a list and therefore, the appropriate NIF functions will need to be used to get this argument as a usable array in C. The *input_bytes* would just be *const int* types and can easily be obtained from the arguments passed. The values passed would also be a list and treated similarly to the *input_dims*.

  ELIXIR FUNCTION

  ```
  create_input_tensor(input_dims, values, input_bytes)
  ```

  C BACKEND CODE

  ```
  TF_Tensor* input = TF_NewTensor(TF_FLOAT, input_dims, input_dims.size(), values, input_bytes, &deallocate, 0);
  inputs = g_array_new (FALSE, FALSE, sizeof (TF_Tensor*));
  g_array_append_val (inputs, input);
  ```

- For the output tensor, as the value will not be populated till the session is run, *TF_AllocateTensor* will be used instead of *TF_NewTensor*:

  ELIXIR FUNCTION

  ```
  create_output_tensor(output_dims, output_bytes)
  ```

  C BACKEND CODE

  ```
  TF_Tensor* output = TF_AllocateTensor(TF_FLOAT, output_dims, 2, output_bytes);
  outputs = g_array_new (FALSE, FALSE, sizeof (TF_Tensor*));
  g_array_append_val (outputs, output);
  ```

- Finally, it is required to run the session and obtain predictions. We will have to pass the previously created *graph*, *inputs*, *outputs*, *input_ops* and *output_ops* arrays as arguments to this function. This will look something like this:

  ELIXIR FUNCTION
  create_and_run_sess(graph, inputs, outputs, input_ops, output_ops)

  C BACKEND CODE
  TF Status* status = TF_NewStatus();
  TF_SessionOptions* sess_opts = TF_NewSessionOptions();
  TF_Session* session = TF_NewSession(graph, sess_opts, status);
  TF_SessionRun(session, 0,
  &inputs[0], &input_values[0], inputs.size(),
  &outputs[0], &output_values[0], outputs.size(),
  0, 0, 0, status);

This gives us a good idea of what TensorflEx will be like for users who want to quickly get a predictions for data. The above example will be implemented by writing NIFs and by allocating resources to different C API structs such as tensors, buffers, graphs, etc. It is also important to understand that functions will have to be general in nature. Something like a *create_tensor* function will have to work irrespective of whether a *char list*, or a *list* or a *float* is passed to it. This is also easily doable using the NIFs library using functions such as *enif_is_list, enif_is_number* and *enif_is_binary*, etc. These functions can check the passed arguments to see which data type it actually is. Therefore, a simple *if* statement will suffice while writing the functions that need to be generalized.

-> LOW-LEVEL OVERVIEW OF TENSORFLEX:

As it would not make sense to list out all the functions in the C API that will be ported as is in TensorflEx, I will list out only some of the major ones. This will also cover the functions listed in the Google guidelines. Moreover, as an example of how these would look like in terms of code, the POC written by me [14], has a few of these implemented. The low-level functions will be as follows:

- **GRAPHS:** *TF_NewGraph, TF_GraphNextOperation, TF_GraphImportGraphDef, TF_GraphOperationByName:*

  TF_NewGraph has been written in the POC and works. TF_GraphNextOperation is easy to implement as it just requires iterating using a for loop in C. The iterations are basically the number of operations in the graph and can be obtained as an integer argument passed through the Elixir function. TF_GraphOperationByName is even easier, as we just need to get the name of

the operation needed to be obtained. TF_GraphImportGraphDef will be implemented by having resources for Buffers and the pre-trained .pb file can be loaded into a graph as is. The implementation for this also falls into the high-level function category discussed above.

- **OPERATIONS:** *TF_NewOperation, TF_FinishOperation:*

  These are also easy to implement. TF_NewOperation has already been implemented and TF_FinishOperation can be implemented by passing in the operation description of a previously created operation using TF_NewOperation.

- **SESSIONS:** *TF_NewSession, TF_RunSession:*

  TF_NewSession and TF_RunSession have not yet been implemented in the general sense. In the POC, a combination of the two which both creates and runs a session for string constant tensors. These C API functions can be implemented generally using NIFs once the create tensor function becomes general in nature. Resources will have to be allocated to Sessions by creating a Session Resource Type. These will have to be deallocated as well. Currently in the POC, resource types are only of the graph type and the operation (description) type.

- **TENSORS:** *TF_Tensor, TF_TensorType:*

  Generalized tensors can be created using a create_tensor function. The Tensor will reflect all the datatypes in TF_DataType. The arguments passed as the value to the tensor can be checked using functions from the NIF library (*enif_is_list, enif_is_number,* etc.) and then be created appropriately depending on the datatype.

- **MISCELLANEOUS:** *TF_Version, TF_DataTypeSize, TF_NewStatus, TF_SetStatus, TF_GetCode, TF_Message, TF_NewBuffer, TF_NewBufferFromString, TF_SetAttrInt, TF_SetAttrType, TF_SetAttrTensor, TF_AddInput:*

  These miscellaneous functions will also be implemented. Most of the aforementioned functions are integral to constructing graphs so these will be useful for programmers who would like to do this in TensorflEx. Their

implementations will not be too difficult to code.

NIFs will be used to write the TensorflEx functions. Currently the POC contains working code written using the NIFs. I have a lot of experience writing C code and will also be referencing the excellent documentation [15] for completing the project.

IV. CURRENT WORK AND PROOF OF CONCEPT (POC):

I have written a POC [14] that uses the Tensorflow C API and allows users to create graphs, populate them with “Const” operations and subsequently run a session. The user can create string constant tensors, run a session with them as input and then obtain the output tensor containing the value of the input tensor. This is essentially the equivalent of the Hello World Tensorflow program in Python. The POC uses Mix, NIFs and the Tensorflow C API to achieve all this. The code can be observed here: https://github.com/anshuman23/tensorflex

V. PERSONAL STATEMENT:

I believe I am a suitable candidate for this project and can lead it to fruitful completion over the summer. I will be working on this full-time and will give weekly updates of the progress on the work. The project requires being comfortable with writing C code and how the Tensorflow C API functions. I had been introduced to C in high school for my first course on data structures and have since been writing C code. As for machine learning, I have been working on research utilizing applied machine learning for the past 2 years and have mainly used Tensorflow and Keras for these projects.

Some of my other relevant achievements are:

- Offered a fully-funded PhD position at the University of California, Davis starting September 2018, after completing my Bachelor’s in India.
- Offered a research internship as an undergraduate at ESnet, Lawrence Berkeley National Laboratory (out of Master's and PhD students as well) for summer 2017. This resulted in a publication at IEEE/ACM Supercomputing 2017
- Offered a 6 week remote internship at Facebook HQ, to work on their Open source networking project-- Warp-speed Data Transfer (wdt) under the guidance of Facebook Engineers starting April 2018
- Sponsored by my university to present two research papers at the 51st IEEE Conference on Information Sciences and Systems (CISS) 2017, held at Johns Hopkins University in March 2017

Working as a GSoC candidate on this project under the guidance of the accomplished Elixir mentors will be a great learning experience for me and will improve my programming abilities manifold. I am positive some constructive work will be done as a

result. Thank you for considering me for this program!

REFERENCES

[1] https://www.tensorflow.org/install/install_c
[2] https://keras.io
[3] https://www.tensorflow.org/versions/master/install/install_go
[4] https://github.com/migueldeicaza/TensorFlowSharp/tree/master/TensorFlowSharp
[5] https://github.com/PerfectlySoft/Perfect-TensorFlow
[6] https://www.tensorflow.org/extend/language_bindings#recommended_approach
[7]https://github.com/migueldeicaza/TensorFlowSharp/blob/master/TensorFlowSharp/Tensor.cs#L475-L499
[8]https://github.com/migueldeicaza/TensorFlowSharp/blob/master/TensorFlowSharp/Tensor.cs#L682-L687
[9]https://github.com/migueldeicaza/TensorFlowSharp/blob/master/TensorFlowSharp/Tensorflow.cs#L269-L272
[10]https://github.com/migueldeicaza/TensorFlowSharp/blob/master/TensorFlowSharp/Tensorflow.cs#L549-564
[11]https://github.com/migueldeicaza/TensorFlowSharp/blob/master/TensorFlowSharp/Tensorflow.cs#L475-L479
[12]https://github.com/PerfectlySoft/Perfect-TensorFlow/blob/master/Sources/TensorFlowAPI/TensorFlowAPI.c
[13] https://developer.gnome.org/glib/stable/glib-Arrays.html
[14] https://github.com/anshuman23/tensorflex
[15] http://erlang.org/doc/man/erl_nif.html