An Analytical Learning on Tensor flow Open Source Software

Saiyam[1], Dr. Raman Chadha[2], Abhishek Kaushik[3]
[13]B. Tech (CSE) 2nd Year, CGC Technical Campus, Jhanjeri, Mohali, India
[2]Professor, head (CSE), CGC Technical Campus, Jhanjeri, Mohali, India

Abstract- Tensor Flow is an open-source software library developed by Google Brain team for carrying out high performance numerical computations using data flow graphs. It provides a great support for implementing machine learning and deep learning models. It was released under the Apache 2.0 open-source license on November 9, 2015.

Keywords- TensorFlow, Machine Learning, Toolkit, Workflow, Applications.

I. INTRODUCTION

TensorFlow is an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. Tensor Flow is flow of data(or tensor) in a computational graph. We can say tensors are multi-dimensional arrays that allows you to represent data having higher dimensions (features of dataset) and the flow means series of operations, that a neural network performs on that data(or tensor).Example: Matrix multiplication can be considered as a numerical computation in which matrix of any dimension refer to as a tensor or data and flow is the operation (multiplication) performed on that tensor.

1. TENSOR(MATRIX)

2	4	3
0	9	5
1	6	7

Now in computational graph, this tensor act as an edge and operation(multiplication) act as a node and thus in this data flow various algorithms can be applied to compute the problem.

2. Why we use it?

Now everyone should question " why should we use tensor flow"when we have so many better frameworks available which focus especially on machine learning models only like -

- Caffe
- MXNet
- Theano
- Torch

The reason is TensorFlow's flexible system architecture that allows computation with high performance on any GPU or CPU, be it a desktop, a server or even a mobile device, irrespective of their computation powers.

- Also it allows you to write your frontend in java, c++ or python. It then takes your code and computes it using its distributed engine and then you are able to run it anywhere.

-It offers advantage over advanced support on threads and asynchronous computations.

- It offers monitoring for training processes of the models and visualization (Tensor-board).

Now we know why should we use tensor flow but one should also know what is machine learning and how can we implement it.

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

II. TYPES OF MACHINE LEARNING

Types of machine learning are:

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

1. Supervised learning : algorithm can apply what has been learned in the past to new data using labeled examples to predict future events. E.g.: Regression , Classification ,Decision tree,Random forest.

2. Unsupervised learning : The model learns through observation and finds structures in the data. Once the model is given a dataset, it automatically finds patterns and relationships in the dataset by creating clusters in it. E.g. Clustering, Association analysis.

3. Reinforcement learning: It is the ability of an agent to interact with the environment and find out what is best outcome. It follows the concept of hit and trial method. Now we have some basic perception about machine learning so let's implement it using tensor flow.

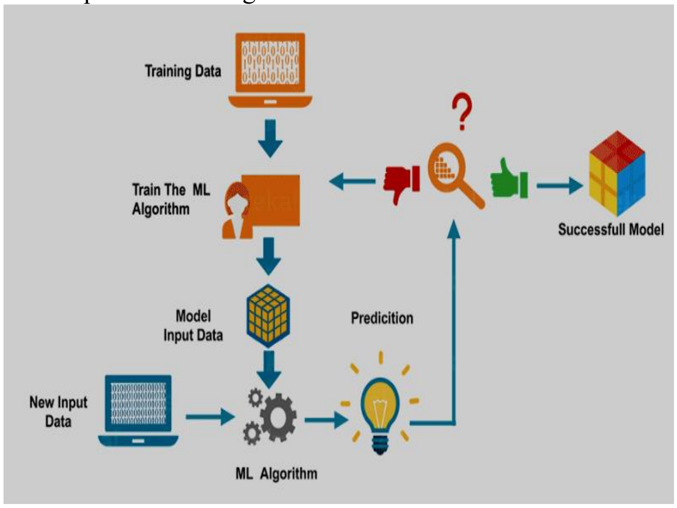
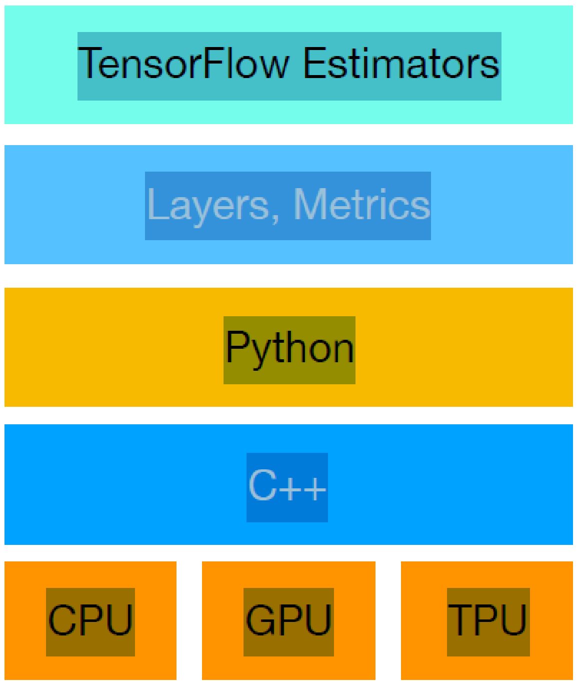


Fig.1: How Machine learning Works

III. TENSORFLOW TOOLKIT

TensorFlow provides a variety of different toolkits that allow you to construct models at your preferred level of abstraction. You can use lower-level APIs to build models by defining a series of mathematical operations. Alternatively, you can use higher-level APIs (like estimator) to specify predefined architectures, such as linear repressors' or neural networks.



We will be using Estimators for now because using estimator dramatically lowers the number of lines of code. It is generally much easier to create models with Estimators than with the low-level TensorFlow APIs. Estimator is a **high-level TensorFlow API** that greatly simplifies machine learning programming. They simplify sharing implementations between model developers. For users who just want to use the common models, TensorFlow provides pre-made estimators or "**Canned Estimators**" which refer to implementations of common machine learning models. Estimators encapsulate the following actions:

• training
• evaluation
• prediction
• export for serving

We will be discussing on creating estimators later but for now lets understand machine learning workflow.

IV. MACHINE LEARNING WORKFLOW

To develop and manage a production-ready model, you must work through the following stages:

• Source and prepare your data.
• Develop your model.
• Train an ML model on your data:
• Train model
• Evaluate model accuracy
• Tune hyper parameters
• Deploy your trained model.
• Send prediction requests to your model:
• Online prediction
• Batch prediction
• Monitor the predictions on an ongoing basis.
• Manage your models and model versions.

Before you start thinking about how to solve a problem with ML, take some time to think about the problem you are trying to solve and then proceed with the above steps.

1. Source and prepare your data

You must have access to a large set of training data that includes the attribute (called a feature in ML) that you want to be able to infer (predict) based on the other features. For example, assume you want your model to predict the sale price of a house. Begin with a large set of data describing the characteristics of houses in a given area, including the sale price of each house.

Data analysis

After sourcing the data, you must analyze and understand the data and prepare it to be the input to the training process.

• Identify features in your data. Features comprise the subset of data attributes that you use in your model.
• Clean the data to find any anomalous values caused by errors in data entry or measurement.

2. Code your model

In this step, you will develop your model and train it on your data using **Estimator API**.

To do this, you need to perform these following steps:

• Build your Estimator model.

Define how data is fed into the model for both training and test datasets (often these definitions are essentially the same).

• Define training and evaluation specifications (TrainSpec and EvalSpec) to be passed to Estimator API. The EvalSpec can include information on how to export your trained model for prediction (serving).

Create an Estimator -

- **Estimators**, which represent a complete model. The Estimator API provides methods to train the model, to judge the model's accuracy, and to generate predictions.
- **Datasets for Estimators**, which build a data input pipeline. The Dataset API has methods to load and manipulate data, and feed it into your model. The Dataset API meshes well with the Estimators API.

Create input functions :

You must create input functions to supply data for training, evaluating, and prediction. An input function is a function that returns a tf. data. Dataset object which outputs the following two-element tuple:

• Features - A Python dictionary in which each key is the name of a feature.
• Label - An array containing the values of the label for every example.

```
def input_evaluation_set():
features = {'SepalLength': np.array([6.4, 5.0]),
'SepalWidth': np.array([2.8, 2.3]),
'PetalLength': np.array([5.6, 3.3]),
'PetalWidth': np.array([2.2, 1.0])}
```

labels = np.array([2, 1])
return features, labels
def train_input_fn(features, labels, batch_size):
"""An input function for training"""
# Convert the inputs to a Dataset. dataset = tf.data.Dataset.from_tensor_slices((dict(features), labels)) # Shuffle, repeat, and batch the examples. Return dataset.shuffle(1000).repeat().batch(batch_size) Here features represent the properties of flowers containing data about them in a numpy array and label is the variable you want to predict. so firstly, we have shown the implementation of input function and then it is defined for training.

Define the feature columns: A feature column is an object describing how the model should use raw input data from the features dictionary. When you build an Estimator model, you pass it a list of feature columns that describes each of the features you want the model to use.

Instantiate an estimator: TensorFlow provides several pre-made classifier Estimators, including:

tf.estimator.DNNClassifier for deep models that perform multi-class classification.

• **tf.estimator.DNNLinearCombinedClassifier** for wide & deep models.

• **tf.estimator.LinearClassifier** for classifiers based on linear models.

You can instantiate using any of the above classifier. Next step is to train and evaluate the model.

Train the model: Train the model by calling the Estimator's **train** method as follows:

# Train the Model.
classifier.train(input_fn=lambda:iris_data.train_input_fn(train_x, train_y, args.batch_size), steps=args.train_steps) Here we wrap up our input_fn call in a **lambda** to capture the arguments while providing an input function that takes no arguments, as expected by the Estimator. The steps argument tells the method to stop training after a number of training steps. Define training and evaluation specifications We just need to define the TrainSpec and EvalSpec used by tf.estimator.train_and_evaluate. These specify not only the input functions, but how to export our trained model; that is, how to save it in the standard SavedModelformat, so that we can later use it for serving.First, we'll define the TrainSpec, which takes as an arg train_input: train_spec = tf.estimator.TrainSpec (train_input, max_steps=1000) For our EvalSpec, we'll instantiate it with something additional – a list of exporters, that specify how to export (save) the trained model so that it can be used for serving with respect to a particular data input format. exporter = tf.estimator.FinalExporter('census', json_serving_input_fn) eval_spec = tf.estimator.EvalSpec(eval_input, steps=100, exporters=[exporter], name='census-eval') We now have a trained model that produces good evaluation results. You can also tune the model by changing the operations or settings that you use to control the training process, such as the number of training steps to run. This technique is known as hyper parameter tuning.

V. MODEL TESTING

During training, you apply the model to known data to adjust the settings to improve the results. When your results are good enough for the needs of your application, you should deploy the model to whatever system your application uses and test it. To test your model, run data through it in a context as close as possible to your final application and your production infrastructure.

Tensor flow in use :

1. RankBrain : Rankbrain is a google's algorithm learning artificial intelligence system.google uses rankbrain algorithm to better its search result.
2. Deep speech : Deep speech is introduced by Mozilla. Deep speech algorithm is used for automatic speech recognition, which aims to make speech technologies and trained models openly available to developers.
3. SmartReply : SmartReply is introduced by google. It is used to automatically generate email response.

VI. REFERENCES

[1]. https://cloud.google.com/mlengine/docs/tensorflow/mlsolutions overview

[2]. https://www.tensorflow.org/guide/premade_estimators

[3]. https://developers.google.com/machinelearning/crashcourse/first-steps-with-tensorflow/toolkit

[4]. https://www.tutorialspoint.com/machine_learning_ with_python/index.htm

[5]. https://www.toptal.com/machine-learning/machine-learningtheory an-introductory-primer

[6]. https://data-flair.training/blogs/machine-learning-tutorial/