Image Recognition using TensorFlow

Domantas Meidus Alessandro Pomes

October 18, 2017

Abstract

The report would show a small part of the huge world of Machine Learning. In particular how we can handle Neural Network with Library of Tensorflow. We provide also an implementation with python that can recognized digit handmade written using famous dataset called MNIST.

1 Introduction

The area of machine learning is rapidly growing up. Artificial intelligent is keeping more and more importance in research world and even if in Commercial area. The huge informatics company have with one of first goal to improve their Sector in AI to provide at their consumers some powerful tools to analyze data.

1.1 Tensorflow history and motivation

ML and Deep learning have been an huge impact in Computer Science area, to allow exploration of new frontier and to push the research through incredible tools that every day a lot of people can use.
Former project of machine learning of Google, a deep learning for internal use, was called: DistBelief. Was developed in 2011 and allowed to Google to bild much more complex Neural Network and have helped to manage the company Datacenter. It was also used to detect some concept from YouTube video and to improve the vocal Recognition of Google more than 25%. DistDistBelief has also trained the inception model that win the Large Scale recognition Challenge in 2014 and allowed to do experiments in the areas of the recognition images.

Altough Distbelif had a lots of succes he had some trouble and limits. It infact was developed and bilt only with neural network, was tricky to configure and was too much linked to the internal structure of Google making it impossible to handle outside of the Company.
Now the new generation of Google Ml is tensorflow and was developed on purpose to cover these misses. Tensorflow is Flexible and easy to use and more than twice faster than DistBelif. Google Brain team was able to represents computations as stateful dataflow graphs. TensorFlow is able to model computations on a wide variety of hardware, from consumer devices such as those powered by Android, to large-scale heterogeneous, multiple GPU systems. TensorFlow claims to be able to, without significant alteration of code, move execution of the computationally expensive tasks of a given graph from solely CPU to heterogeneous GPU-accelerated environments. Given these details, it goes without saying that TensorFlow aims to bring massive parallelism and high scalability to machine learning for all.

2 Tensors

Very briefly, a tensor is an N-dimensional array containing the same type of data (int32, bool, etc.): All you need to describe a tensor fully is its data type and the value of each of the N dimension.

That's why we describe a tensor with what we call a shape: it is a list, tuple or TensorShape of numbers containing the size of each dimension of our tensor, for example:

- for a tensor of n dimensions: $(D0, D1, \ldots, Dn - 1)$
- a tensor of size W x H: (W, H)
- for a tensor of size W (usually called a vector): $(W,)$
- For a simple scalar (those are equivalent): $()or(1,)$

some different type of Tensor we can we can define in the code are:

- tf.Variable
- tf.Constant
- tf.Placeholder
- tf.SparseTensor

With the exception of tf.Variable, the value of a tensor is immutable, which means that in the context of a single execution tensors only have a single value. However, evaluating the same tensor twice can return different values. For example that tensor can be the result of reading data from disk, or generating a random number.

3 Graph and sessions

Everything in TensorFlow is based on creating a computational graph. We can think of a computational graph as a network of nodes, with each node known as an operation, running some function that can be as simple as addition or subtraction to as complex as some multi variate equation.

An Operation also referred to as "op" can return zero or more tensors which can be used later on in the graph.

Each operation can be handed a tensor,for instance, as we seen above, 2-dimensional tensor is equivalent to a m x m matrix, so we can for example compute very easily multiplication with matrix. All inputs needed by the "op" are run automatically. They're typically ran in parallel. The "session run" actually causes the execution of the operations in the graph. So we can describe in some point the directed, acyclic graph in:

- Bottom nodes (no arrows going in) are inputs (external inputs (data) or internal inputs (variables))
- Top nodes (no arrows going out) are outputs
- Middle nodes are functions

4 MNIST

4.1 description of the Dataset

- An immage handmade written Xs
- A corresponding label Ys. Labbels of the digits that goes from 0 to 9
- Data are divided in 3 parts: 55'000 points of training (mnist.train) 10'000 points of test (mnist.test) and 5'000 points of convalidation (mnist.validation)

4.2 Jupyter Notebook and implementation

Setup

Import tensorflow and read MNIST dataset

```python
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

Placeholder

Value that we'll input when we ask TensorFlow to run a computation.

None

Dimension can be of any length.

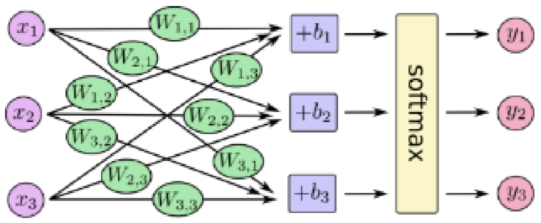
784

We want to be able to input any number of MNIST images, each flattened into a 784-dimensional vector.

```python
x = tf.placeholder(tf.float32, [None, 784])
```

We also need the weights and biases for our model. We could imagine treating these like additional inputs, but TensorFlow has an even better way to handle it: Variable. A Variable is a modifiable tensor that lives in TensorFlow's graph of interacting operations. It can be used and even modified by the computation. For machine learning applications, one generally has the model parameters be Variables.

```python
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
```



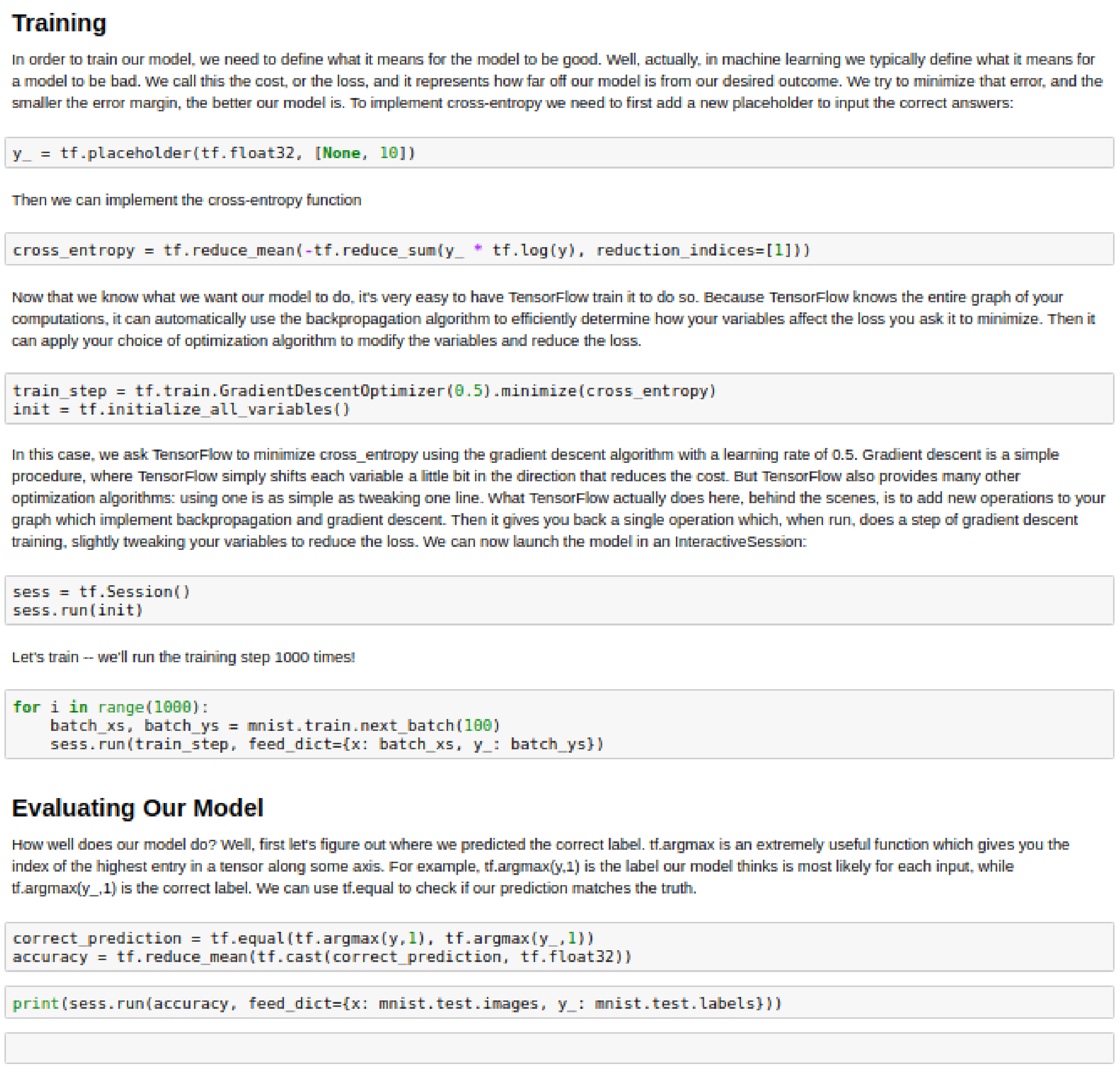
If we write that out as equations, we get:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \begin{bmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{bmatrix}$$

We can "vectorize" this procedure, turning it into a matrix multiplication and vector addition. This is helpful for computational efficiency. (It's also a useful way to think.)

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

```python
y = tf.nn.softmax(tf.matmul(x, W) + b)
```



Training

In order to train our model, we need to define what it means for the model to be good. Well, actually, in machine learning we typically define what it means for a model to be bad. We call this the cost, or the loss, and it represents how far off our model is from our desired outcome. We try to minimize that error, and the smaller the error margin, the better our model is. To implement cross-entropy we need to first add a new placeholder to input the correct answers:

```
y_ = tf.placeholder(tf.float32, [None, 10])
```

Then we can implement the cross-entropy function

```
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
```

Now that we know what we want our model to do, it's very easy to have TensorFlow train it to do so. Because TensorFlow knows the entire graph of your computations, it can automatically use the backpropagation algorithm to efficiently determine how your variables affect the loss you ask it to minimize. Then it can apply your choice of optimization algorithm to modify the variables and reduce the loss.

```
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
init = tf.initialize_all_variables()
```

In this case, we ask TensorFlow to minimize cross_entropy using the gradient descent algorithm with a learning rate of 0.5. Gradient descent is a simple procedure, where TensorFlow simply shifts each variable a little bit in the direction that reduces the cost. But TensorFlow also provides many other optimization algorithms: using one is as simple as tweaking one line. What TensorFlow actually does here, behind the scenes, is to add new operations to your graph which implement backpropagation and gradient descent. Then it gives you back a single operation which, when run, does a step of gradient descent training, slightly tweaking your variables to reduce the loss. We can now launch the model in an InteractiveSession:

```
sess = tf.Session()
sess.run(init)
```

Let's train -- we'll run the training step 1000 times!

```
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

Evaluating Our Model

How well does our model do? Well, first let's figure out where we predicted the correct label. tf.argmax is an extremely useful function which gives you the index of the highest entry in a tensor along some axis. For example, tf.argmax(y,1) is the label our model thinks is most likely for each input, while tf.argmax(y_,1) is the correct label. We can use tf.equal to check if our prediction matches the truth.

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

```
print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

References

1. https://www.tensorflow.org/

2. TensorFlow - Google's latest machine learning system, open sourced for everyone Monday, November 09, 2015 Posted by Jeff Dean, Senior Google Fellow, and Rajat Monga, Technical Lea

3. http://yann.lecun.com/exdb/mnist/

4. http://jupyter.org/documentation.html