

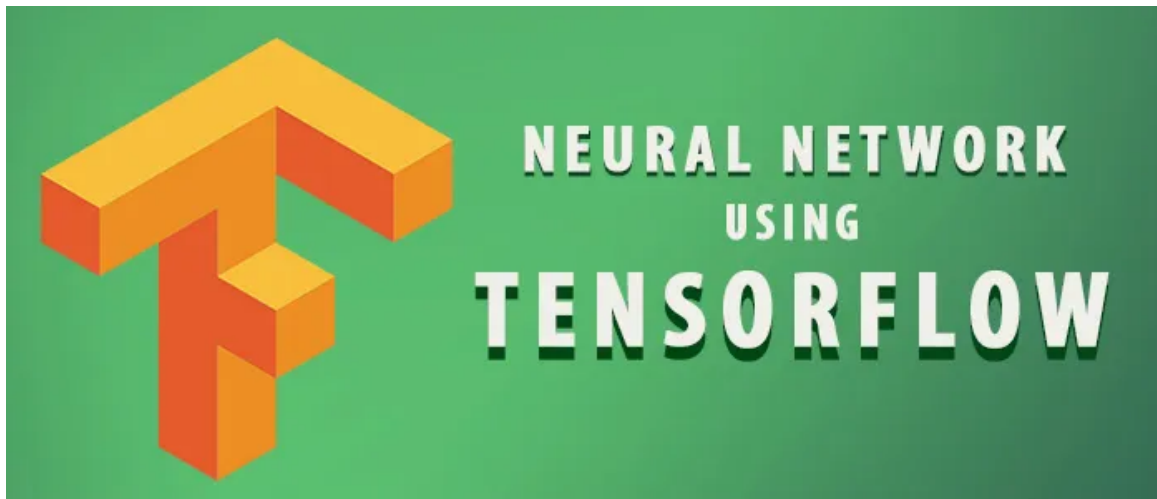
# A Developer Diary

{about:"code learn and share"}

[Home](#)[Data Science](#)[Java](#)[JavaScript](#)[jBPM](#)[Tools](#)[Tips](#)[About](#)

April 19, 2019 By [Abhisek Jana](#) — [Leave a Comment](#) ([Edit](#))

## Implement Neural Network using TensorFlow



In the previous article we have implemented the Neural Network using Python from scratch. However for real implementation we mostly use a framework, which generally provides faster computation and better support for best practices. In this article we will Implement Neural Network using TensorFlow. At present, TensorFlow probably is the most popular deep learning framework available.

Here is my previous post on “Understand and Implement the Backpropagation Algorithm From Scratch In Python”. We will be implementing the similar example here using TensorFlow. In case you need a refresher please refer the article below:



## Understand and Implement the Backpropagation Algorithm From Scratch In Python

It's very important have clear understanding on how to implement a simple Neural Network from scratch. In this Understand and Implement the Backpropagation Algorithm From Scratch In Python tutorial we go through step by step process of understanding and implementing a Neural Network. We will start from Linear Regression and use the same concept to ... [Continue reading](#)



A Developer Diary



24



## Notes on TensorFlow:

There are few points worth to understand about TensorFlow which is different from other frameworks or our previous implementation.

- TensorFlow has been written to keep production deployment in mind. I would say exactly opposite to R, which is very much focused on analysis and study. Hence, there are few design considerations which need to understand.
- TensorFlow mainly supports Static Computation Graph ( However there will be support for Dynamic Computation Graph in version 2. You might have already heard of Eager Execution.) In this example we will work with Static Computation Graph. The main problem with Static Computation Graph is support for debugging. So if you make a mistake you can really use Python editor to perform line by line debugging.
- In past years, TensorFlow went through many many design changes. Hence you might find many different ways of creating Neural Network using TensorFlow. I will be using the most recent recommendations. In case you google you will find many variations of the same code,

however be careful with the version they have used, since the code might already be outdated.

- TensorFlow is currently integrating **Keras** as High Level API. If you visit the TensorFlow website you will find plenty of example using Keras.
- In real production implementation system generally you will always save the model and run prediction by loading the model separately. In short, you generally save the model after training and load it during testing or real uses. There is no way to save the model as instance variable in Python. Since we just want to focus on building the Network, we will run train and test both inside the **fit()** function using the same session. You will generally never do this in real production system.

## Dataset:

We will be using the MNIST dataset. It has 60K training images, each **28X28** pixel in gray scale. There are total 10 classes to classify. You can find more details about it in the following sites:

[https://en.wikipedia.org/wiki/MNIST\\_database](https://en.wikipedia.org/wiki/MNIST_database)

<http://yann.lecun.com/exdb/mnist/index.html>

## Coding using TensorFlow:

We will create a class named **ANN** and define the following functions. As discussed earlier, the **fit\_predict()** function will train our model and then run the prediction on the test data using the same session.

As you have noticed we will pass the layers needed for our network as a list so that we don't have to code them explicitly.

```
model = ANN(layers_size=[196, 10])
model.fit_predict(train_x, train_y, test_x, test_y, learning_rate=0.1,
n_iterations=1000)
model.plot_cost()
```

### **\_\_init\_\_()** function:

Start by defining the **\_\_init\_\_()** method. The **self.parameters** and **self.store** dictionary will be used to save the computed values during **forward()** so that these values can be reused.

```
def __init__(self, layers_size):
    self.costs = []
```

```

self.layers_size = layers_size
self.parameters = {}
self.L = len(layers_size)
self.store = {}
self.X = None
self.Y = None

```

## initialize\_parameters() function:

The `initialize_parameters()` function will be used to initialize the `W` and `b` parameters for our Network.

If we already know number of layers and hidden units, we can simply define them as following. However defining like this way will not help if we want to try out more layers with different hidden units.

```

W1 = tf.get_variable("W1", shape=[196, f],
initializer=tf.contrib.layers.xavier_initializer(seed=1))
b1 = tf.get_variable("b1", shape=[196, 1],
initializer=tf.zeros_initializer())
W2 = tf.get_variable("W2", shape=[10, 196],
initializer=tf.contrib.layers.xavier_initializer(seed=1))
b2 = tf.get_variable("b2", shape=[10, 1],
initializer=tf.zeros_initializer())

```

Hence we will dynamically define them by looping through the `self.layers_size` list.

We will use `get_variable()` function which is a relatively latest addition to TensorFlow where we can define a supported initializer. Here we will be using `xavier_initializer` for `W` and `zeros_initializer` for `b`.

**Note:** In case you do not know what's `Xavier Initializer`, don't worry about it. I will later make another tutorial on it.

```

def initialize_parameters(self):
    tf.set_random_seed(1)

    for l in range(1, self.L + 1):
        self.parameters["W" + str(l)] = tf.get_variable("W" + str(l),
                                                         shape=

```

```
[self.layers_size[l], self.layers_size[l - 1]],

initializer=tf.contrib.layers.xavier_initializer(seed=1))
    self.parameters["b" + str(l)] = tf.get_variable("b" + str(l),
shape=[self.layers_size[l], 1],

initializer=tf.zeros_initializer())
```

## forward() function:

Next let's define the `forward()` function.

We can code for the fixed number of layers like following.

```
Z1 = tf.add(tf.matmul(W1, tf.transpose(self.X)), b1)
A1 = tf.nn.relu(Z1)
Z2 = tf.add(tf.matmul(W3, A2), b3)
```

However, it will be wise to dynamically perform the forward propagation. Few points to be noted,

- We are using `ReLU` Activation function.
- When  $l=1$ ,  $A^{[0]}$  will be equal to `X`
- For all layers, calculate and store the  $Z^{[l]}$  in memory inside the loop.
- For all layers from `l=1 to L-1`, calculate and store the  $A^{[l]}$  in memory inside the loop.
- Since final layer will use Softmax activation, we will use TensorFlow's builtin function `softmax_cross_entropy_with_logits_v2()`.

```
def forward(self):
    for l in range(1, len(self.layers_size)):
        if l == 1:
            self.store["Z" + str(l)] =
tf.add(tf.matmul(self.parameters["W" + str(l)], tf.transpose(self.X)),

self.parameters["b" + str(l)])
        else:
            self.store["Z" + str(l)] = tf.add(
                tf.matmul(self.parameters["W" +
str(l)], self.store["A" + str(l - 1)]),
                self.parameters["b" + str(l)])
```

```

        if l < self.L:
            self.store["A" + str(l)] =
tf.nn.relu(self.store["Z" + str(l)])

        softmax =
tf.nn.softmax_cross_entropy_with_logits_v2(logits=tf.transpose(self.store["Z"
+ str(self.L)]), labels=self.Y)
        return softmax

```

## fit\_predict() function:

TensorFlow will automatically calculate the derivatives for us, hence the backpropagation will be just a like of code. Lets go through the `fit_predict()` function.

First we will find the number of features from the shape of `X_train` and the number of classes from the shape of `Y`. The shape of `X_train` in our example here is `(60000, 784)` and The shape of `Y_train` is `(60000, 10)`.

Then we will define two placeholder `X, Y` based on number of features and classes.

We will insert the number of features in our `layers_size` list since technically the input layer is layer 0. We will need the size to define the `W1`.

Finally we will call `self.initialize_parameters()` and `self.forward()` function.

```

tf.set_random_seed(1)
_, f = X_train.shape
_, c = Y_train.shape

self.X = tf.placeholder(tf.float32, shape=[None, f], name='X')
self.Y = tf.placeholder(tf.float32, shape=[None, c], name='Y')

self.layers_size.insert(0, f)

self.initialize_parameters()

softmax = self.forward()

```

Next we will define our cost function and then use TensorFlow's builtin function for Gradient Descent Optimization. Feel free to try out other optimization functions available. The `minimize()` function will help to calculate all the derivatives with respect to the cost function.

```
cost = tf.reduce_mean(softmax)
optimizer =
tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)
```

We are done with defining our static computation graph. We have not feed the data into our model yet. Lets do that next by creating a TensorFlow session.

We need to call `global_variables_initializer()` for TensorFlow's Global Variable Initialization. Next we will create a Session and loop through the `n_iterations`.

Inside the loop we will have TensorFlow compute the `optimizer` and `cost` variable. At this point we need to pass the data using the `feed_dict` parameter.

```
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    for epoch in range(n_iterations):
        _, epoch_cost = sess.run([optimizer, cost], feed_dict=
{self.X: X_train, self.Y: Y_train})
```

We want to calculate the Train accuracy in every 100 iteration and also save the cost in every 10 iteration. The below code is very simple, we will compare the predicted values with target variable. Then find the accuracy by calling `reduce_mean()` function.

```
if epoch % 100 == 0:
    correct_prediction = tf.equal(tf.argmax(self.store["Z" +
str(self.L)]),

tf.argmax(tf.transpose(self.Y)))

    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    print("Cost after epoch %i: %f, Accuracy %f" % (
        epoch, epoch_cost, accuracy.eval({self.X: X_train,
self.Y: Y_train})))

if epoch % 10 == 0:
    self.costs.append(epoch_cost)
```

Once training is complete we will calculate the accuracy of the test data inside the same session.

```

correct_prediction = tf.equal(tf.argmax(self.store["Z" +
str(self.L)]),tf.argmax(tf.transpose(self.Y)))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
print("Test Accuracy %f" % (accuracy.eval({self.X: X_test, self.Y:
Y_test})))

```

## \_\_main\_\_():

Finally let's look at our `main()` method. First we will get the data. Then we will preprocess it. Afterwards, call the `fit_predict()` function of the `ANN()` class.

```

if __name__ == '__main__':
    train_x_orig, train_y_orig, test_x_orig, test_y_orig =
mnist.get_data()

    train_x, train_y, test_x, test_y = pre_process_data(train_x_orig,
train_y_orig, test_x_orig, test_y_orig)

    print("train_x's shape: " + str(train_x.shape))
    print("test_x's shape: " + str(test_x.shape))

    model = ANN(layers_size=[196, 10])
    model.fit_predict(train_x, train_y, test_x, test_y,
learning_rate=0.1, n_iterations=1000)
    model.plot_cost()

```

## pre\_process\_data():

In the preprocessing step we will first normalize the data by dividing by 255. Then we will use `OneHotEncoder` of the `sklearn` package to transform the target variable.

```

def pre_process_data(train_x, train_y, test_x, test_y):
    # Normalize
    train_x = train_x / 255.
    test_x = test_x / 255.

    enc = OneHotEncoder(sparse=False, categories='auto')
    train_y = enc.fit_transform(train_y.reshape(len(train_y), -1))
    test_y = enc.transform(test_y.reshape(len(test_y), -1))

```



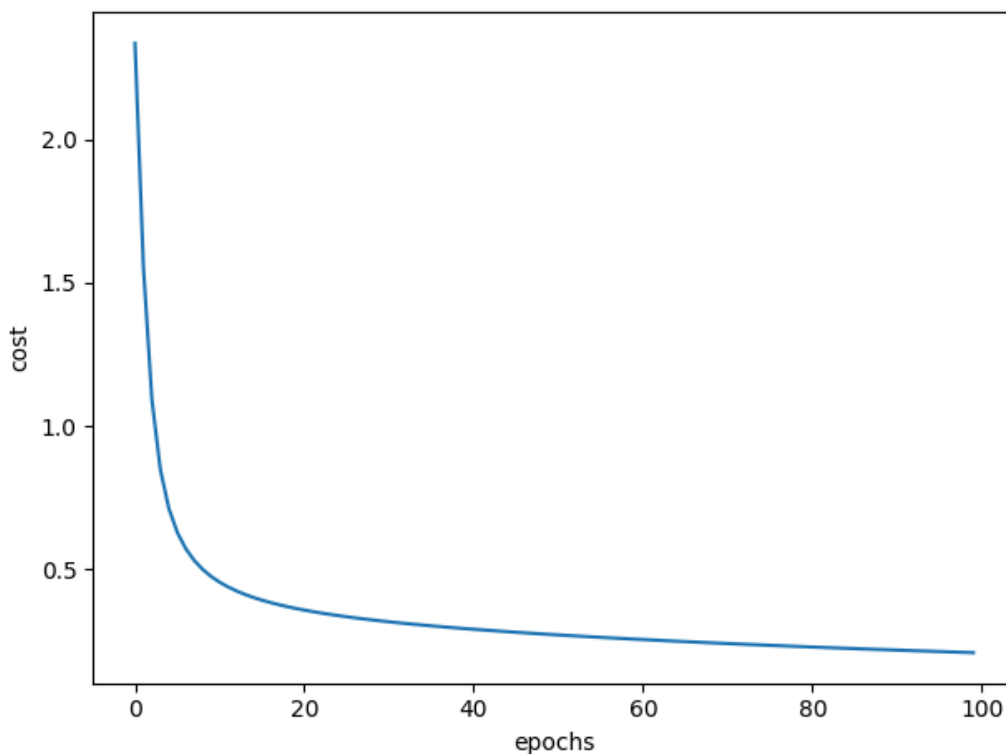
```
return train_x, train_y, test_x, test_y
```

## Output:

Now its time to run the our code. With just a 2-Layer Network and 1000 epoch we are getting around 94% of accuracy.

```
train_x's shape: (60000, 784)
test_x's shape: (10000, 784)
Cost after epoch 0: 2.336267, Accuracy 0.192117
Cost after epoch 100: 0.455633, Accuracy 0.881817
Cost after epoch 200: 0.358374, Accuracy 0.901867
Cost after epoch 300: 0.317593, Accuracy 0.911667
Cost after epoch 400: 0.291452, Accuracy 0.918683
Cost after epoch 500: 0.271708, Accuracy 0.924483
Cost after epoch 600: 0.255534, Accuracy 0.929317
Cost after epoch 700: 0.241680, Accuracy 0.932850
Cost after epoch 800: 0.229520, Accuracy 0.936000
Cost after epoch 900: 0.218720, Accuracy 0.939083
Test Accuracy 0.942200
```

Here is the plot of the cost function.



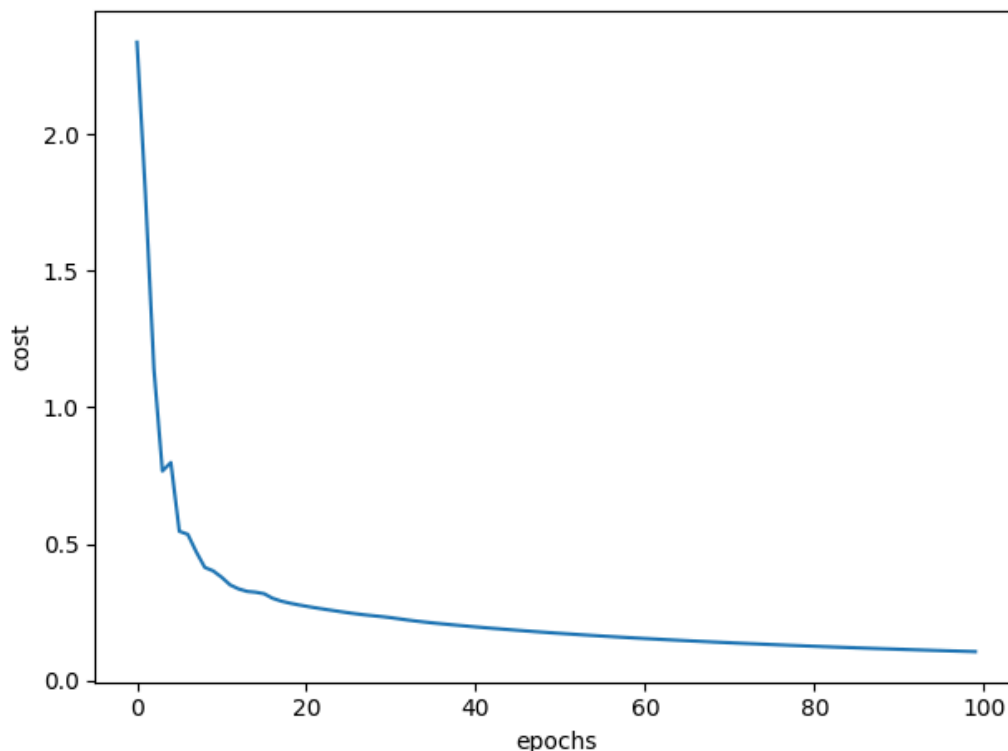
You can try using different Network Layout such as:

```
layers_dims = [392,196,98,1]
```

Here is the result. Our accuracy increased to 96%.

```
train_x's shape: (60000, 784)
test_x's shape: (10000, 784)
Cost after epoch 0: 2.338752, Accuracy 0.187967
Cost after epoch 100: 0.377190, Accuracy 0.893700
Cost after epoch 200: 0.270761, Accuracy 0.922600
Cost after epoch 300: 0.228978, Accuracy 0.934517
Cost after epoch 400: 0.195207, Accuracy 0.944633
Cost after epoch 500: 0.171445, Accuracy 0.951867
Cost after epoch 600: 0.152584, Accuracy 0.956917
Cost after epoch 700: 0.137145, Accuracy 0.961433
Cost after epoch 800: 0.124189, Accuracy 0.965167
Cost after epoch 900: 0.113151, Accuracy 0.968233
Test Accuracy 0.964000
```

Here is the plot of the cost function.



Below is the full code of the ANN class.

```

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import datasets.mnist.loader as mnist
from sklearn.preprocessing import OneHotEncoder

class ANN:
    def __init__(self, layers_size):
        self.costs = []
        self.layers_size = layers_size
        self.parameters = {}
        self.L = len(layers_size)
        self.store = {}
        self.X = None
        self.Y = None

    def initialize_parameters(self):
        tf.set_random_seed(1)

        for l in range(1, self.L + 1):
            self.parameters["W" + str(l)] = tf.get_variable("W" +
str(l),
                                                                    shape=
[self.layers_size[l], self.layers_size[l - 1]],

initializer=tf.contrib.layers.xavier_initializer(seed=1))
            self.parameters["b" + str(l)] = tf.get_variable("b" +
str(l), shape=[self.layers_size[l], 1],

initializer=tf.zeros_initializer())

    def forward(self):
        for l in range(1, len(self.layers_size)):

            if l == 1:
                self.store["Z" + str(l)] =
tf.add(tf.matmul(self.parameters["W" + str(l)], tf.transpose(self.X)),
                                                                    self.parameters["b" +
str(l)])
            else:

```

```

        self.store["Z" + str(l)] = tf.add(
            tf.matmul(self.parameters["W" + str(l)],
self.store["A" + str(l - 1)]),
            self.parameters["b" + str(l)])
    if l < self.L:
        self.store["A" + str(l)] = tf.nn.relu(self.store["Z" +
str(l)])

    softmax =
tf.nn.softmax_cross_entropy_with_logits_v2(logits=tf.transpose(self.store["Z"
+ str(self.L)]),

labels=self.Y)

    return softmax

def fit_predict(self, X_train, Y_train, X_test, Y_test,
learning_rate=0.01, n_iterations=2500):
    tf.set_random_seed(1)
    _, f = X_train.shape
    _, c = Y_train.shape

    self.X = tf.placeholder(tf.float32, shape=[None, f], name='X')
    self.Y = tf.placeholder(tf.float32, shape=[None, c], name='Y')

    self.layers_size.insert(0, f)

    self.initialize_parameters()

    softmax = self.forward()

    cost = tf.reduce_mean(softmax)

    optimizer =
tf.train.GradientDescentOptimizer(learning_rate=learning_rate).minimize(cost)

    init = tf.global_variables_initializer()

    with tf.Session() as sess:
        sess.run(init)

```

```

        for epoch in range(n_iterations):
            _, epoch_cost = sess.run([optimizer, cost], feed_dict=
{self.X: X_train, self.Y: Y_train})

            if epoch % 100 == 0:
                correct_prediction =
tf.equal(tf.argmax(self.store["Z" + str(self.L)]),
tf.argmax(tf.transpose(self.Y)))

                accuracy =
tf.reduce_mean(tf.cast(correct_prediction, "float"))
                print("Cost after epoch %i: %f, Accuracy %f" % (
                    epoch, epoch_cost, accuracy.eval({self.X:
X_train, self.Y: Y_train})))

            if epoch % 10 == 0:
                self.costs.append(epoch_cost)

                correct_prediction = tf.equal(tf.argmax(self.store["Z" +
str(self.L)]),
tf.argmax(tf.transpose(self.Y)))

                accuracy = tf.reduce_mean(tf.cast(correct_prediction,
"float"))
                print("Test Accuracy %f" % (accuracy.eval({self.X: X_test,
self.Y: Y_test})))

        def plot_cost(self):
            plt.figure()
            plt.plot(np.arange(len(self.costs)), self.costs)
            plt.xlabel("epochs")
            plt.ylabel("cost")
            plt.show()

        def pre_process_data(train_x, train_y, test_x, test_y):
            # Normalize
            train_x = train_x / 255.

```

```

test_x = test_x / 255.

enc = OneHotEncoder(sparse=False, categories='auto')
train_y = enc.fit_transform(train_y.reshape(len(train_y), -1))
test_y = enc.transform(test_y.reshape(len(test_y), -1))

return train_x, train_y, test_x, test_y

if __name__ == '__main__':
    train_x_orig, train_y_orig, test_x_orig, test_y_orig =
mnist.get_data()

    train_x, train_y, test_x, test_y = pre_process_data(train_x_orig,
train_y_orig, test_x_orig, test_y_orig)

    print("train_x's shape: " + str(train_x.shape))
    print("test_x's shape: " + str(test_x.shape))

    model = ANN(layers_size=[196, 10])
    model.fit_predict(train_x, train_y, test_x, test_y,
learning_rate=0.1, n_iterations=1000)
    model.plot_cost()

```

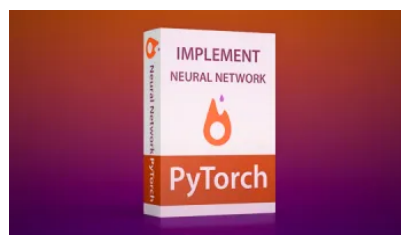
You can access the full project here:

Code

## Related



Understanding and implementing  
Neural Network with SoftMax in  
Python from scratch



Implement Neural Network using  
PyTorch  
In "Data Science"



Understand and Implement the  
Backpropagation Algorithm From  
Scratch In Python

---

Filed Under: [Data Science](#), [Deep Learning](#)

Tagged With: [Deep Learning](#), [Machine Learning](#), [MNIST](#), [Neural Network](#), [Python](#), [TensorFlow](#)

## Subscribe to stay in loop

\* indicates required


Email Address \*

Subscribe

## Leave a Reply

Logged in as Abhisek Jana. [Edit your profile](#). [Log out?](#) Required fields are marked \*

Comment \*



Post Comment

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

Copyright © 2024 A Developer Diary