

Tensorflow Example: Fizzbuzz

Sargur N. Srihari
srihari@cedar.buffalo.edu

Fizzbuzz in Tensorflow

- Fizzbuzz:
 - Print the numbers from 1 to 100, except that if the number is divisible by 3 print "fizz", if it's divisible by 5 print "buzz", and if it's divisible by 15 print "fizzbuzz"
- We look at implementation of Fizzbuzz in Tensorflow
 - As a simple MLP with one hidden layer
 - See <http://joelgrus.com/2016/05/23/fizz-buzz-in-tensorflow/>

Useful Python syntax

1. `%` operator, e.g., `b % a`
 - divides left-hand operator by right-hand operator and returns remainder
2. `==` condition, e.g., `a==b`
 - Condition becomes true if operands are equal
3. `&`, e.g., `a&b`
 - Bitwise logical AND operator
4. Range function, e.g., `range(3)=[0,1,2]`

Standard Imports

- `import numpy as np`
- `import tensorflow as tf`

Input treated as a vector

- Input is a number, output is "fizzbuzz" representation of that number.
- In particular, we need to turn each input into a vector of "activations". One simple way would be to convert it to binary.
 - Define a binary encoder for the input i

```
def binary_encode(i, num_digits):  
    return np.array([i >> d & 1 for d in range(num_digits)])
```

Mapping input to fizzbuzz

```
def fizz_buzz_encode(i):  
    if i % 15 == 0: return np.array([0, 0, 0, 1])  
    elif i % 5 == 0: return np.array([0, 0, 1, 0])  
    elif i % 3 == 0: return np.array([0, 1, 0, 0]) else:  
return np.array([1, 0, 0, 0])
```

Generating Training Samples

- It would be cheating to use the numbers 1 to 100 in our training data, so let's train it on all the remaining numbers up to 1024:

```
NUM_DIGITS = 10
```

```
trX = np.array([binary_encode(i, NUM_DIGITS)  
for i in range(101, 2 ** NUM_DIGITS)])
```

```
trY = np.array([fizz_buzz_encode(i)  
for i in range(101, 2 ** NUM_DIGITS)])
```

No. of Hidden Units

- Chosen arbitrarily to be 10

NUM_HIDDEN = 100

Input and Output Variables

- We'll need an input variable of with width NUM_DIGITS
- Output variable with width 4

```
X = tf.placeholder("float", [None, NUM_DIGITS])
```

```
Y = tf.placeholder("float", [None, 4])
```

Randomly initialized weights

- One hidden layer and one output layer

```
def init_weights(shape):  
    return tf.Variable(tf.random_normal(shape,  
        stddev=0.01))  
  
w_h = init_weights([NUM_DIGITS, NUM_HIDDEN])  
w_o = init_weights([NUM_HIDDEN, 4])
```

ReLU Activation

- Ready to define model using ReLU activation

```
def model(X, w_h, w_o):  
    h = tf.nn.relu(tf.matmul(X, w_h))  
    return tf.matmul(h, w_o)
```

Softmax Cross-Entropy Cost

- We try and minimize it

```
py_x = model(X, w_h, w_o)
```

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(py_x,  
Y))
```

```
train_op = tf.train.GradientDescentOptimizer(0.05).minimize(cost)
```

Prediction

- Prediction will just be the largest output

```
predict_op = tf.argmax(py_x, 1)
```

- `predict_op` will output a number from 0 to 3 but we want a fizzbuzz output

```
def fizz_buzz(i, prediction):
```

```
    return [str(i), "fizz", "buzz", "fizzbuzz"][prediction]
```

Training

- We grab a tensorflow session and initialize the variables

```
with tf.Session() as sess:
```

```
    tf.initialize_all_variables().run()
```

Training epochs

- 10,000 epochs of training
- Shuffle them for each iteration

for epoch in range(10000):

```
p = np.random.permutation(range(len(trX)))  
trX, trY = trX[p], trY[p]
```

Each Training Pass

- Each epoch trained in batches of 128

BATCH_SIZE=128

- Each training pass looks like

```
for start in range(0, len(trX), BATCH_SIZE):  
    end = start + BATCH_SIZE  
    sess.run(train_op,  
    feed_dict={X: trX[start:end], Y: trY[start:end]})
```


Printing Accuracy of Training Data

- It is helpful see how training accuracy evolves

```
print(epoch, np.mean(np.argmax(trY, axis=1) ==  
    sess.run(predict_op, feed_dict={X: trX, Y:  
    trY})))
```

Fizzbuzz Testing

- Input is just the binary encoding of numbers 1 to 100

```
numbers = np.arange(1, 101)
```

```
teX = np.transpose(binary_encode(numbers,  
NUM_DIGITS))
```

Output

- Output is fizzbuzz function applied to model output

```
teY = sess.run(predict_op, feed_dict={X: teX})  
output = np.vectorize(fizz_buzz)(numbers, teY)  
print(output)
```

Performance

- In [185]: output
- Out[185]:
- `array(['1', '2', 'fizz', '4', 'buzz', 'fizz', '7', '8', 'fizz', 'buzz', '11', 'fizz', '13', '14', 'fizzbuzz', '16', '17', 'fizz', '19', 'buzz', '21', '22', '23', 'fizz', 'buzz', '26', 'fizz', '28', '29', 'fizzbuzz', '31', 'fizz', 'fizz', '34', 'buzz', 'fizz', '37', '38', 'fizz', 'buzz', '41', '42', '43', '44', 'fizzbuzz', '46', '47', 'fizz', '49', 'buzz', 'fizz', '52', 'fizz', 'fizz', 'buzz', '56', 'fizz', '58', '59', 'fizzbuzz', '61', '62', 'fizz', '64', 'buzz', 'fizz', '67', '68', '69', 'buzz', '71', 'fizz', '73', '74', 'fizzbuzz', '76', '77', 'fizz', '79', 'buzz', '81', '82', '83', '84', 'buzz', '86', '87', '88', '89', 'fizzbuzz', '91', '92', '93', '94', 'buzz', 'fizz', '97', '98', 'fizz', 'fizz'],`
- `dtype='<U8')`

Conclusion

- Running this code on GitHub got some of the outputs wrong!
 - I count 0.90 fizz-accuracy, and 0.99 buzz-accuracy. So it's clearly harder to teach fizzing than buzzing.
- A deeper network may help