CSE 5523: Lecture Notes 18
Automatic Differentiation

Contents

Building and debugging Jacobians for backpropagation in neural networks can be time consuming.

But as we've seen, Jacobians are usually cleanly associated with individual matrix operations.

This means it's possible to just write cost functions and let the compiler figure out your Jacobians!

Packages that do this are called **automatic differentiation** or **autodiff**.

Here are some examples using **tensorflow** in python, which works well with numpy and pandas:

18.1 Sample multi-layer neural network code

Here's our multi-layer neural network coded in tensorflow:

```
import sys
import numpy
import pandas
import tensorflow as tf

YX = pandas.read_csv( sys.argv[1] )                    ## read in data
Y = pandas.get_dummies( YX[YX.columns[0]] )            ## transform data
X = YX[YX.columns[1:]]
N = len(YX)
X['line'] = numpy.ones((N,1))

X0 = tf.constant( X.T , dtype=tf.float32 )             ## tensorflow format
Y0 = tf.constant( Y.T , dtype=tf.float32 )
print( X0 )
print( Y0 )
                                                       ## trainable matrices
W1 = tf.Variable( tf.random.uniform( [5,len(X.columns)], dtype=tf.float32  ), trainable=True )
W2 = tf.Variable( tf.random.uniform( [len(Y.columns),5], dtype=tf.float32  ), trainable=True )

def cost():                                            ## cost function
  H1   = tf.nn.softmax( W1 @ X0, axis=0 )
  Yhat = tf.nn.softmax( W2 @ H1, axis=0 )
  return - 1/N * tf.reduce_sum( tf.math.log( tf.reduce_sum( Y0 * Yhat, axis=0 ) ) )

opt = tf.keras.optimizers.SGD( 1. )                    ## optimization algorithm
```

```
for epoch in range( 100 ):                              ## for each epoch
  opt.minimize( cost, var_list=[W1,W2] )                ## minimize cost

print( W1 )
print( W2 )

print( tf.nn.softmax( W2 @ tf.nn.softmax( W1 @ X0, axis=0 ), axis=0 ) )
print( cost() )
```

Sample input data file 'YX.csv':

```
y,x1,x2
ya,-1,-1
ya,-1,1
ya,1,-1
ya,1,1
ya,0,0
no,-2,-2
no,-2,2
no,2,-2
no,2,2
```

Output trained weights and predictions (my newlines):

```
tf.Tensor(
[[-1. -1.  1.  1.  0. -2. -2.  2.  2.]
 [-1.  1. -1.  1.  0. -2.  2. -2.  2.]
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]], shape=(3, 9), dtype=float32)

tf.Tensor(
[[0. 0. 0. 0. 0. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1. 0. 0. 0. 0.]], shape=(2, 9), dtype=float32)

<tf.Variable 'Variable:0' shape=(5, 3) dtype=float32, numpy=
array([[-2.4127858 ,  0.5126198 , -0.63468486],
       [ 0.5846569 ,  0.5350458 ,  0.32132182],
       [ 0.61990684,  0.49538755,  0.19627862],
       [ 0.5839069 ,  0.5125111 ,  3.3187284 ],
       [ 3.5522873 ,  0.51258224, -0.5775441 ]], dtype=float32)>

<tf.Variable 'Variable:0' shape=(2, 5) dtype=float32, numpy=
array([[ 2.3994162 , -0.2641866 , -0.17200986, -2.2945478 ,  2.277045  ],
       [-1.3906401 ,  1.5341104 ,  1.4361174 ,  3.4277508 , -1.4699322 ]],
      dtype=float32)>

tf.Tensor(
[[0.0477829 0.04782108 0.0497507 0.04977838 0.0063128 0.9340164 0.9339868 0.93148446 0.9314576]
 [0.9522171 0.95217896 0.9502493 0.95022166 0.9936871 0.0659835 0.0660131 0.06851552 0.0685423]],
shape=(2, 9), dtype=float32)

tf.Tensor(0.05388033, shape=(), dtype=float32)
```

18.2 Sample recurrent neural network code

Here's our recurrent neural network coded in tensorflow – note backprop in for loop:

```
import sys
import numpy
import pandas
import tensorflow as tf

YX = pandas.read_csv( sys.argv[1] )                          ## read in data
Y = pandas.get_dummies( YX[YX.columns[0]] )                  ## transform data
X = YX[YX.columns[1:]]
N = len(YX)
X['line'] = numpy.ones((N,1))

X0 = tf.constant( X, dtype=tf.float32 )                      ## tensorflow format
Y0 = tf.constant( Y, dtype=tf.float32 )
                                                             ## trainable matrices
W_H = tf.Variable( tf.random.uniform( [5,5+len(X.columns)], dtype=tf.float32 ), trainable=True )
W_Y = tf.Variable( tf.random.uniform( [len(Y.columns),5],   dtype=tf.float32 ), trainable=True )

def estimate():                                              ## calculate estimate
  Yhat = []                                                  ## store output in list
  h = tf.Variable( numpy.zeros((5,1)), dtype=tf.float32 )    ## initial hidden state
  for t in range(len(YX)):                                   ## iterate over input
    xT = tf.reshape( X0[t], (len(X.columns),1) )             ## update hidden state
    h  = tf.nn.softmax( W_H @ tf.concat( [h,xT], axis=0 ), axis=0 )
    Yhat.append( tf.reshape( tf.nn.softmax( W_Y @ h, axis=0 ), (len(Y.columns),) ) )
  return tf.stack(Yhat)                                      ## output as matrix

def cost():                                                  ## cross entropy cost
  return - 1/N * tf.reduce_sum( tf.math.log( tf.reduce_sum( Y0 * estimate(), axis=1 ) ) )

opt = tf.keras.optimizers.SGD( 1. )                          ## specify SGD optimizer

for epoch in range( 100 ):                                   ## for each epoch
  opt.minimize( cost, var_list=[W_H,W_Y] )                   ## do fwd and backprop

print( W_H )                                                 ## output models
print( W_Y )

print( estimate() )                                          ## output estimate
print( cost() )                                              ## output final cost
```

Sample input data file 'YX.csv':

```
y,x
ya,1
ya,1
ya,1
no,0
no,0
no,1
no,0
```

```
no,1
no,0
no,0
```

Output trained weights and predictions (my newlines):

```
<tf.Variable 'Variable:0' shape=(5, 7) dtype=float32, numpy=
array([[ 3.063445  ,  0.74077016,  0.34251413,  0.35408616,  1.2225189 , -1.2436845 ,  2.5310872 ],
       [ 0.03648799,  0.6387872 ,  0.22822534,  0.5413278 ,  0.25724894,  0.0848136 , -0.307701  ],
       [-0.21812092,  0.52940553,  0.07440143,  0.28256175,  0.15348482,  0.9460333 , -0.40608412],
       [-1.309914  , -0.01481871,  0.7782022 ,  0.73846525,  0.40991116,  2.5245852 ,  1.1286684 ],
       [ 0.70981693,  0.04093007,  0.5038499 ,  0.42033607,  0.25866374, -0.07557707, -0.3658372 ]],
dtype=float32)>

<tf.Variable 'Variable:0' shape=(2, 5) dtype=float32, numpy=
array([[ 3.228355  ,  0.72423214,  0.41444364, -1.2663867 ,  1.1153417 ],
       [-1.9325942 ,  0.62781274,  1.039471  ,  2.7991576 ,  0.41628927]],
      dtype=float32)>

tf.Tensor(
[[0.04487731 0.9551227 ]
 [0.04424097 0.955759  ]
 [0.0446894  0.9553106 ]
 [0.95614773 0.04385229]
 [0.99310946 0.00689052]
 [0.97777474 0.02222525]
 [0.9934691  0.00653085]
 [0.97854644 0.0214535 ]
 [0.9934822  0.00651783]
 [0.99381196 0.00618796]], shape=(10, 2), dtype=float32)

tf.Tensor(0.025210267, shape=(), dtype=float32)
```