COSC4337_DynamicRNN

[1]:
```
#Dynamic Recurrent Neural Network.

#TensorFlow implementation of a Recurrent Neural Network (LSTM) that performs
#dynamic computation over sequences with variable length. This example is using
#a toy dataset to classify linear sequences. The generated sequences have
#variable length.

#Links:
#    [Long Short Term Memory](http://deeplearning.cs.cmu.edu/pdfs/
 ↪Hochreiter97_lstm.pdf)


from __future__ import print_function

import tensorflow as tf
import random


# ====================
#  TOY DATA GENERATOR
# ====================
class ToySequenceData(object):
    """ Generate sequence of data with dynamic length.
    This class generate samples for training:
    - Class 0: linear sequences (i.e. [0, 1, 2, 3,...])
    - Class 1: random sequences (i.e. [1, 3, 10, 7,...])

    NOTICE:
    We have to pad each sequence to reach 'max_seq_len' for TensorFlow
    consistency (we cannot feed a numpy array with inconsistent
    dimensions). The dynamic calculation will then be perform thanks to
    'seqlen' attribute that records every actual sequence length.
    """
    def __init__(self, n_samples=1000, max_seq_len=20, min_seq_len=3,
                 max_value=1000):
        self.data = []
        self.labels = []
        self.seqlen = []
        for i in range(n_samples):
            # Random sequence length
            len = random.randint(min_seq_len, max_seq_len)
            # Monitor sequence length for TensorFlow dynamic calculation
            self.seqlen.append(len)
            # Add a random or linear int sequence (50% prob)
            if random.random() < .5:
                # Generate a linear sequence
                rand_start = random.randint(0, max_value - len)
                s = [[float(i)/max_value] for i in
                     range(rand_start, rand_start + len)]
                # Pad sequence for dimension consistency
                s += [[0.] for i in range(max_seq_len - len)]
                self.data.append(s)
                self.labels.append([1., 0.])
            else:
                # Generate a random sequence
                s = [[float(random.randint(0, max_value))/max_value]
                     for i in range(len)]
                # Pad sequence for dimension consistency
                s += [[0.] for i in range(max_seq_len - len)]
                self.data.append(s)
                self.labels.append([0., 1.])
        self.batch_id = 0

    def next(self, batch_size):
        """ Return a batch of data. When dataset end is reached, start over.
        """
        if self.batch_id == len(self.data):
            self.batch_id = 0
        batch_data = (self.data[self.batch_id:min(self.batch_id +
                                                  batch_size, len(self.data))])
        batch_labels = (self.labels[self.batch_id:min(self.batch_id +
                                                  batch_size, len(self.data))])
        batch_seqlen = (self.seqlen[self.batch_id:min(self.batch_id +
                                                  batch_size, len(self.data))])
        self.batch_id = min(self.batch_id + batch_size, len(self.data))
        return batch_data, batch_labels, batch_seqlen


# ==========
#   MODEL
# ==========

# Parameters
learning_rate = 0.01
```

```
training_steps = 10000
batch_size = 128
display_step = 200

# Network Parameters
seq_max_len = 20 # Sequence max length
n_hidden = 64 # hidden layer num of features
n_classes = 2 # linear sequence or not

trainset = ToySequenceData(n_samples=1000, max_seq_len=seq_max_len)
testset = ToySequenceData(n_samples=500, max_seq_len=seq_max_len)

# tf Graph input
x = tf.placeholder("float", [None, seq_max_len, 1])
y = tf.placeholder("float", [None, n_classes])
# A placeholder for indicating each sequence length
seqlen = tf.placeholder(tf.int32, [None])

# Define weights
weights = {
    'out': tf.Variable(tf.random_normal([n_hidden, n_classes]))
}
biases = {
    'out': tf.Variable(tf.random_normal([n_classes]))
}


def dynamicRNN(x, seqlen, weights, biases):

    # Prepare data shape to match `rnn` function requirements
    # Current data input shape: (batch_size, n_steps, n_input)
    # Required shape: 'n_steps' tensors list of shape (batch_size, n_input)

    # Unstack to get a list of 'n_steps' tensors of shape (batch_size, n_input)
    x = tf.unstack(x, seq_max_len, 1)

    # Define a lstm cell with tensorflow
    lstm_cell = tf.contrib.rnn.BasicLSTMCell(n_hidden)

    # Get lstm cell output, providing 'sequence_length' will perform dynamic
    # calculation.
    outputs, states = tf.contrib.rnn.static_rnn(lstm_cell, x, dtype=tf.float32,
                                sequence_length=seqlen)

    # When performing dynamic calculation, we must retrieve the last
    # dynamically computed output, i.e., if a sequence length is 10, we need
    # to retrieve the 10th output.
    # However TensorFlow doesn't support advanced indexing yet, so we build
    # a custom op that for each sample in batch size, get its length and
    # get the corresponding relevant output.

    # 'outputs' is a list of output at every timestep, we pack them in a Tensor
    # and change back dimension to [batch_size, n_step, n_input]
    outputs = tf.stack(outputs)
    outputs = tf.transpose(outputs, [1, 0, 2])

    # Hack to build the indexing and retrieve the right output.
    batch_size = tf.shape(outputs)[0]
    # Start indices for each sample
    index = tf.range(0, batch_size) * seq_max_len + (seqlen - 1)
    # Indexing
    outputs = tf.gather(tf.reshape(outputs, [-1, n_hidden]), index)

    # Linear activation, using outputs computed above
    return tf.matmul(outputs, weights['out']) + biases['out']

pred = dynamicRNN(x, seqlen, weights, biases)

# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred,␣
 ↪labels=y))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate).
 ↪minimize(cost)

# Evaluate model
correct_pred = tf.equal(tf.argmax(pred,1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:

    # Run the initializer
    sess.run(init)

    for step in range(1, training_steps + 1):
        batch_x, batch_y, batch_seqlen = trainset.next(batch_size)
        # Run optimization op (backprop)
        sess.run(optimizer, feed_dict={x: batch_x, y: batch_y,
                                       seqlen: batch_seqlen})
        if step % display_step == 0 or step == 1:
            # Calculate batch accuracy & loss
            acc, loss = sess.run([accuracy, cost], feed_dict={x: batch_x, y:␣
↪batch_y,
                                                              seqlen: batch_seqlen})
            print("Step " + str(step*batch_size) + ", Minibatch Loss= " + \
                  "{:.6f}".format(loss) + ", Training Accuracy= " + \
                  "{:.5f}".format(acc))

    print("Optimization Finished!")

    # Calculate accuracy
    test_data = testset.data
    test_label = testset.labels
    test_seqlen = testset.seqlen
    print("Testing Accuracy:", \
        sess.run(accuracy, feed_dict={x: test_data, y: test_label,
                                      seqlen: test_seqlen}))
```

```
WARNING:tensorflow:
The TensorFlow contrib module will not be included in TensorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-contrib-
sunset.md
  * https://github.com/tensorflow/addons
  * https://github.com/tensorflow/io (for I/O related ops)
If you depend on functionality not listed there, please file an issue.

WARNING:tensorflow:From <ipython-input-1-13bf36a7acaa>:121:
BasicLSTMCell.__init__ (from tensorflow.python.ops.rnn_cell_impl) is deprecated
and will be removed in a future version.
Instructions for updating:
This class is equivalent as tf.keras.layers.LSTMCell, and will be replaced by
that in Tensorflow 2.0.
WARNING:tensorflow:From <ipython-input-1-13bf36a7acaa>:126: static_rnn (from
tensorflow.python.ops.rnn) is deprecated and will be removed in a future
version.
Instructions for updating:
Please use `keras.layers.RNN(cell, unroll=True)`, which is equivalent to this
API
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\python\ops\rnn_cell_impl.py:735: Layer.add_variable
(from tensorflow.python.keras.engine.base_layer) is deprecated and will be
removed in a future version.
Instructions for updating:
Please use `layer.add_weight` method instead.
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\python\ops\rnn_cell_impl.py:739: calling Zeros.__init__
(from tensorflow.python.ops.init_ops) with dtype is deprecated and will be
removed in a future version.
```

```
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\python\ops\rnn.py:244: where (from
tensorflow.python.ops.array_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From <ipython-input-1-13bf36a7acaa>:153:
softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) is
deprecated and will be removed in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.

Step 128, Minibatch Loss= 0.707822, Training Accuracy= 0.42188
Step 25600, Minibatch Loss= 0.694900, Training Accuracy= 0.51923
Step 51200, Minibatch Loss= 0.692359, Training Accuracy= 0.51923
Step 76800, Minibatch Loss= 0.690161, Training Accuracy= 0.51923
Step 102400, Minibatch Loss= 0.686136, Training Accuracy= 0.51923
Step 128000, Minibatch Loss= 0.676407, Training Accuracy= 0.58654
Step 153600, Minibatch Loss= 0.649035, Training Accuracy= 0.58654
Step 179200, Minibatch Loss= 0.571427, Training Accuracy= 0.72115
Step 204800, Minibatch Loss= 0.508370, Training Accuracy= 0.75962
Step 230400, Minibatch Loss= 0.507132, Training Accuracy= 0.73077
Step 256000, Minibatch Loss= 0.504104, Training Accuracy= 0.74038
Step 281600, Minibatch Loss= 0.500213, Training Accuracy= 0.75962
Step 307200, Minibatch Loss= 0.497013, Training Accuracy= 0.75962
Step 332800, Minibatch Loss= 0.494399, Training Accuracy= 0.75000
Step 358400, Minibatch Loss= 0.492184, Training Accuracy= 0.74038
Step 384000, Minibatch Loss= 0.490240, Training Accuracy= 0.74038
Step 409600, Minibatch Loss= 0.488486, Training Accuracy= 0.72115
Step 435200, Minibatch Loss= 0.486863, Training Accuracy= 0.72115
Step 460800, Minibatch Loss= 0.485329, Training Accuracy= 0.71154
Step 486400, Minibatch Loss= 0.483852, Training Accuracy= 0.71154
Step 512000, Minibatch Loss= 0.482404, Training Accuracy= 0.71154
Step 537600, Minibatch Loss= 0.480960, Training Accuracy= 0.74038
Step 563200, Minibatch Loss= 0.479497, Training Accuracy= 0.74038
Step 588800, Minibatch Loss= 0.477991, Training Accuracy= 0.75000
Step 614400, Minibatch Loss= 0.476413, Training Accuracy= 0.75000
Step 640000, Minibatch Loss= 0.474732, Training Accuracy= 0.75000
Step 665600, Minibatch Loss= 0.472910, Training Accuracy= 0.75000
Step 691200, Minibatch Loss= 0.470899, Training Accuracy= 0.75000
Step 716800, Minibatch Loss= 0.468643, Training Accuracy= 0.75000
Step 742400, Minibatch Loss= 0.466069, Training Accuracy= 0.75962
Step 768000, Minibatch Loss= 0.463080, Training Accuracy= 0.76923
Step 793600, Minibatch Loss= 0.459541, Training Accuracy= 0.76923
Step 819200, Minibatch Loss= 0.455251, Training Accuracy= 0.76923
Step 844800, Minibatch Loss= 0.449885, Training Accuracy= 0.77885
Step 870400, Minibatch Loss= 0.442876, Training Accuracy= 0.77885
Step 896000, Minibatch Loss= 0.433159, Training Accuracy= 0.78846
Step 921600, Minibatch Loss= 0.418573, Training Accuracy= 0.79808
Step 947200, Minibatch Loss= 0.394290, Training Accuracy= 0.80769
Step 972800, Minibatch Loss= 0.364629, Training Accuracy= 0.80769
Step 998400, Minibatch Loss= 0.452984, Training Accuracy= 0.80769
Step 1024000, Minibatch Loss= 0.420527, Training Accuracy= 0.79808
Step 1049600, Minibatch Loss= 0.359778, Training Accuracy= 0.79808
Step 1075200, Minibatch Loss= 0.157666, Training Accuracy= 0.99038
Step 1100800, Minibatch Loss= 0.124582, Training Accuracy= 0.98077
Step 1126400, Minibatch Loss= 0.108477, Training Accuracy= 0.99038
Step 1152000, Minibatch Loss= 0.095143, Training Accuracy= 0.99038
Step 1177600, Minibatch Loss= 0.085568, Training Accuracy= 0.99038
Step 1203200, Minibatch Loss= 0.078872, Training Accuracy= 0.99038
Step 1228800, Minibatch Loss= 0.073785, Training Accuracy= 0.99038
Step 1254400, Minibatch Loss= 0.069655, Training Accuracy= 0.99038
Step 1280000, Minibatch Loss= 0.066146, Training Accuracy= 0.99038
Optimization Finished!
Testing Accuracy: 0.98
```

[ ]: