Chapter_9_Step_By_Step_Example

February 3, 2019

1 Example of convolution nn

1.1 Step 1 - Input reshape

First we reshape our data so they can pass through the `tf.conv2d` function. According the documentation of `tf.conv2d` the input needs to be as `[batch, in_height, in_width, in_channels]`. Since in this case we are doing a simple pass through example with one image we will reshape our input:

```
idx=4 # selected example from data
exampleData = data[idx, :]
# batch=1, in_height=24, in_width=24, in_channels=1 (since it's greyscale)
exampleDataReshaped = tf.reshape(exampleData, shape=[-1, 24, 24, 1])
```

1.2 Step 2 - Convolution

The reshaped image is then passed through the convolution filter:

```
conv = tf.nn.conv2d(exampleDataReshaped, W, strides=[1, 1, 1, 1], padding='SAME')
conv_with_b = tf.nn.bias_add(conv, b)
conv_out = tf.nn.relu(conv_with_b)
```

`W` `b` are `tf` variables that hold the weights to be trained. These define the output shape of `conv_out`. For example if:

```
W = tf.Variable(tf.random_normal([5, 5, 1, 64]))
b = tf.Variable(tf.random_normal([64]))
```

where `[5, 5, 1, 64]`: 5,5 -> size of filter on image, 1-> input dimension (1 for grayscale, 3 for RGB), 64 -> number of convolutions

The result of this step is an array with shape `(1, 24, 24, 64)`. This holds 64 24x24 filtered images that depend on `W` and `b` (which are trainable). These images can be considered image processing features extracted from the original image.

1.3 Step 3 - Max Pool

We can prevent overfit by artifically reducing the numer of parameters of our model. This can be done by taking the max value of 2,3 or more adjacent array elements. This is done using `tf.nn.max_pool`. For example the result of the previous example can be sampled by `tf.nn.max_pool` to half its size:

```
maxPool = tf.nn.max_pool(conv_out, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
```

The result is that the filtered images have now a shape of `(1, 12, 12, 64)`.

1.4 Step 4 - Normalization

The result of step 3 can be normalized using `tf.nn.lrn` which takes the input tensor (shape `(1, 12, 12, 64)`) and treats it as a 3D array of 1D vectors (along the last dimension), and each vector is normalized independently. Within a given vector, each component is divided by the weighted, squared sum of inputs within `depth_radius`.

So if we take one of these vectors they would look like `vec1=maxPool[:,:,:,0], vec2=maxPool[:,:,:,1],...` and these are the filtered images. For each filtered image we perform normalisation as described at https://www.tensorflow.org/api_docs/python/tf/nn/local_response_normalization.

In the example this is performed as:

```
norm = tf.nn.lrn(maxPool, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75)
```

These sort of normalisation techniques seem to have fallen out of favour such as dropout and batch normalization.

1.5 Step 5 - Fully connected layer

Steps 2-4 can be repeated. At the end we will pass the results through a fully connected layer. In the example this operation by:

```
norm_reshaped = tf.reshape(norm, [-1, W2.get_shape().as_list()[0]])
full = tf.add(tf.matmul(norm_reshaped, W2), b2)
full_out = tf.nn.relu(full)
```

`W2` has a shape that matches the flatten dimensions of the ouput of the convolution steps. For example if the output shape of norm is `(1, 12, 12, 64)`. Then `W2` will have a shape of `(1*12*12*64, 1024)` where `1024` is just the output dimensions of the fully connected layer.

NOTE: The variable names are general and DO NOT refer to the example code below

1.6 Step 6 - Final output

Finally we need to produce a result in the same shape as our target (labels), which are one hot encoded. In the example we have we have only one label, since we are passing a single image through our nn model.

```
out = tf.add(tf.matmul(full_out, W_out), b_out)
```

`Wout` has a shape that matches the target. In our example the target is a one-hot encoded vector of 10 different labels so `Wout` has shape `(1024,10)`

NOTE: The variable names are general and DO NOT refer to the example code below

1.7 Step 7 - Cost

To be able to train our nn model we need a cost between the targets (labels) and the output of the model. For this we use the `tf.nn.softmax_cross_entropy_with_logits` function:

```
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=out, labels=target))
```

NOTE: The variable names are general and DO NOT refer to the example code below

1.8 Code

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        import cifar_tools_DC
        import tensorflow as tf

        learning_rate = 0.001

        names, data, labels = \
            cifar_tools_DC.read_data(
                '/media/damianos/New Volume/Desktop/Projects/Machine Learning with Tensorflow/

        # have variables outside of session
        W1 = tf.Variable(tf.random_normal([5, 5, 1, 64]))
        b1 = tf.Variable(tf.random_normal([64]))
        W2 = tf.Variable(tf.random_normal([5, 5, 64, 64]))
        b2 = tf.Variable(tf.random_normal([64]))
        W3 = tf.Variable(tf.random_normal([6*6*64, 1024]))
        b3 = tf.Variable(tf.random_normal([1024]))
        W_out = tf.Variable(tf.random_normal([1024, len(names)]))
        b_out = tf.Variable(tf.random_normal([len(names)]))
        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
            sess.run(tf.local_variables_initializer())
            onehot_labels = tf.one_hot(labels, len(names), on_value=1., off_value=0., axis=-1)
            onehot_labels_eval = sess.run(onehot_labels)
            # select an example image and an example label to process
            exampleData = data[4, :]
            exampleLabel = onehot_labels_eval[4, :]
            # reshape input to be able to pass through tf.conv2d
            exampleDataReshaped = tf.reshape(exampleData, shape=[-1, 24, 24, 1])
            print('shape of input is {}'.format(sess.run(exampleDataReshaped).shape))
            # use a 5x5 filter to produce 64 different filtered images
            conv1 = tf.nn.conv2d(exampleDataReshaped, W1, strides=[1, 1, 1, 1], padding='SAME'
            conv_with_b1 = tf.nn.bias_add(conv1, b1)
            conv_out1 = tf.nn.relu(conv_with_b1)
            print('shape of first convolution is {}'.format(sess.run(conv_out1).shape))
            ## max pool
            k = 2
            max1 = tf.nn.max_pool(conv_out1, ksize=[1, k, k, 1], strides=[1, k, k, 1], padding=
            print('shape of first max pool is {}'.format(sess.run(max1).shape))
            ## normalisation
            norm1 = tf.nn.lrn(max1, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75)
            print('shape of first normalisation is {}'.format(sess.run(norm1).shape))
            # repeat another convolution
            conv2 = tf.nn.conv2d(norm1, W2, strides=[1, 1, 1, 1], padding='SAME')
            conv_with_b2 = tf.nn.bias_add(conv2, b2)
            conv_out2 = tf.nn.relu(conv_with_b2)
            print('shape of second convolution is {}'.format(sess.run(conv_out2).shape))
            ## normalisation
            norm2 = tf.nn.lrn(conv_out2, 4, bias=1.0, alpha=0.001 / 9.0, beta=0.75)
            print('shape of second normalisation is {}'.format(sess.run(norm2).shape))
            ## max pool
            max2 = tf.nn.max_pool(norm2, ksize=[1, k, k, 1], strides=[1, k, k, 1], padding='SA
            print('shape of second max pool is {}'.format(sess.run(max2).shape))
            ## reshape last result (max2) to be passed through a fully connected layer
            ## whose shape equals the flatten shape of max2 and the shape of the output
            ## from the fully connected layer
            max3_reshaped = tf.reshape(max2, [-1, W3.get_shape().as_list()[0]])
            print('shape of third reshaped max pool is {}'.format(sess.run(max3_reshaped).shap
            print('shape of third nn weight array is {}'.format(sess.run(W3).shape))
            local3 = tf.add(tf.matmul(max3_reshaped, W3), b3)
            local_out3 = tf.nn.relu(local3)
            print('shape of third fully connected layer is {}'.format(sess.run(local_out3).sha
            # final output
            out = tf.nn.sigmoid(tf.add(tf.matmul(local_out3, W_out), b_out))
            print('shape of final output is {}'.format(sess.run(out).shape))
            # cost function
            cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=out, labels=e
            print('model output is {}'.format(sess.run(out)))
            print('target is {}'.format(exampleLabel))
            print('cost is {}'.format(sess.run(cost)))
```

```
names ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck
(50000, 3072) (50000,)
shape of input is (1, 24, 24, 1)
shape of first convolution is (1, 24, 24, 64)
shape of first max pool is (1, 12, 12, 64)
shape of first normalisation is (1, 12, 12, 64)
shape of second convolution is (1, 12, 12, 64)
shape of second normalisation is (1, 12, 12, 64)
shape of second max pool is (1, 6, 6, 64)
shape of third reshaped max pool is (1, 2304)
shape of third nn weight array is (2304, 1024)
shape of third fully connected layer is (1, 1024)
shape of final output is (1, 10)
WARNING:tensorflow:From <ipython-input-1-c55c61553504>:67: softmax_cross_entropy_with_logits (
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}.

model output is [[1. 0. 1. 0. 1. 0. 1. 1. 1. 0.]]
target is [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
cost is 3.0110981464385986
```