CONVOLUTIONAL NEURAL NETWORKS (CNN)

Edit

Note: A basic introduction to CNNs and Tensorflow implementation. If anyone wants naive numpy python implementation with backpropagation and beyond, check the GitHub repo.

I. OBJECTIVE:

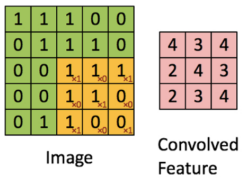
CNNs are traditionally used for processing data that can be convolved over which allows us to encode specific properties into the NN architecture.

Ex: To process a 32X32X3 image with a vanilla NN would require (32*32*3) weights just for the first hidden layer. This full connectivity is inefficient and would quickly lead to many parameters.

CNN architecture:

II. FILTERS:

Filters (kernels, weights, etc.) slide across our image or subsequent layer outputs in order to extract features.



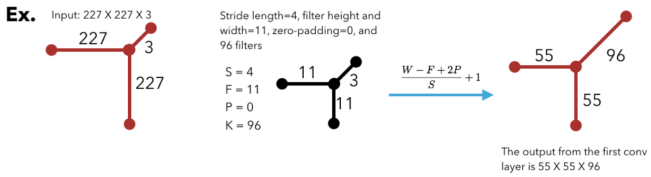
Think of convolution as a sliding window (**filter**) function applied to a matrix.

http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

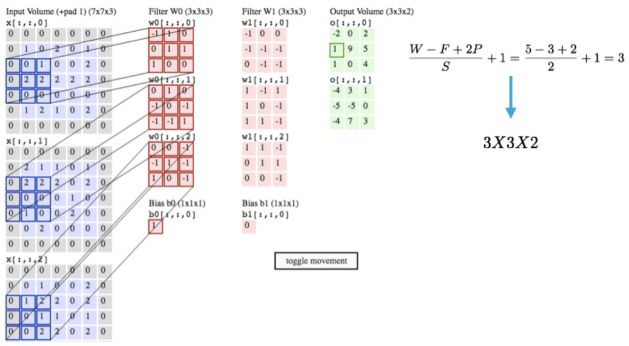
Filter weights are randomly or empirically initialized and are learned to become feature detectors. The filters on the first convolutional layer tend to learn low level features (edges, etc.) and subsequent layers' filters learn high level features.

III. CONV LAYER OUTPUT:

Unlike FC layers, each neuron in the conv layer will only have **local connectivity** to a local region of the whole image.



We can control the output dimensions by controlling zero-padding and stride.



- SEARCH -

Search …

- FOLLOW ME ON TWITTER -

Tweets by @GokuMohandas

Goku Mohandas
@GokuMohandas

Your model may be performing really well by incorrectly focusing on confounding features (extraneous influencers in the data that aren't accounted for). Check out @johnrzech's post where x-ray stickers unintentionally influenced the classifications: medium.com/@jrzech/what-a…

Embed View on Twitter

- RECENT POSTS -

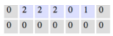
Update on Embeddings (Spring 2017)

Exploring Sparsity in Recurrent Neural Networks

Question Answering from Unstructured Text by Retrieval and Comprehension

Overcoming Catastrophic Forgetting in Neural Networks

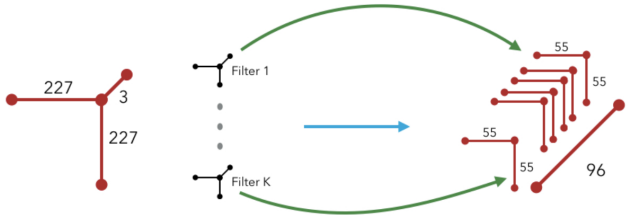
Opening the Black Box of Deep Neural Networks via Information



http://cs231n/github.io/convolutional-networks

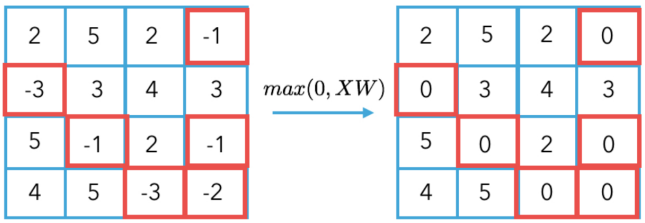
IV. PARAMETER SHARING:

Each filter convolves on the input image and creates a 55 X 55 2D matrix. The K filters together create the output of the first conv layer which is 55 X 55 X K. Each of the 55 X 55 neurons for each depth slice of the output from the first conv layer use the same weights (11 X 11 X 3). The intuition here is that each filter represents a feature and it makes sense to apply the same filter for the entire image using the same weights.



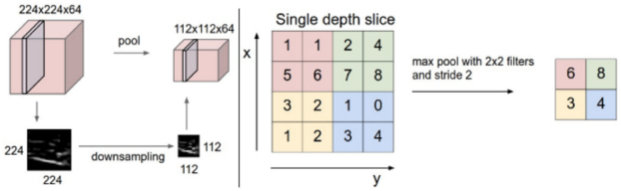
V. RELU UNITS:

ReLU applied at each depth slice (2D output from input and filter i). ReLU performs better than sigmoid and tanh units due to less of an effect from vanishing gradients but they still have pitfalls. For a more comprehensive look at different non-linear units and advantages and disadvantages of each, check out this **post.**



VI. POOLING:

Pooling involves downsampling for each depth slice. Here we see max-pooling, but we can also use other methods (avg. pooling, L2-norm pooling, etc.)



http://cs231n/github.io/convolutional-networks

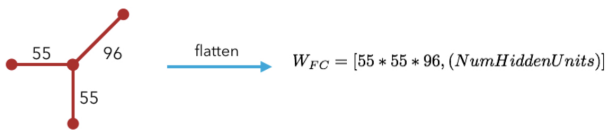
Intuition is to decrease the size of our processing units because we are still able to capture enough of the information needed by pooling.

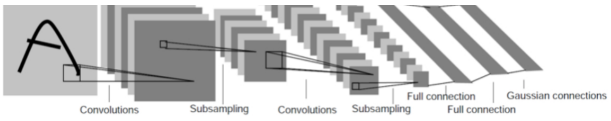
$$W_1 X H_1 X D_1 \longrightarrow W_2 X H_2 X D_2 \quad \text{, where } W_2 = \frac{W_1 - F}{S} + 1$$
$$D_2 = D_1$$

Many current models are learning towards reducing or completely removing the use of pooling since it leads to loss of information. You will start to see clever architectures completely based on convolutional operations with smaller filter sizes.

VII. FC LAYERS:

The final layers of the CNN will be fully-connected (FC) layers, which will flatten the output from the CNN. The final layer will be a classifier that will solve our desired task.





LeNet5

VIII. BACKPROPAGATION:

Backpropagation for the convolution and pooling is slightly convoluted but take a look at the naive python implementation while reading the math for better understanding.

Layer	Forward Pass	Backpropagation
CONV	$z_{(p,q)} = \sum_{u,v} W_{(u,v)} a_{(p+u,q+v)}$	$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial W} = \frac{\partial L}{\partial z_{(p,q)}} a_{(p+u,q+v)} = \frac{\partial L}{\partial z_{(u,v)}} a_{(u,v)}$
ReLU	$y_l = max(0, y_{l-1})$	$\frac{\partial L}{\partial y_{l-1}} = \begin{cases} 0 & \text{if } y_l \leq 0 \\ \frac{\partial L}{\partial y_l} & \text{otherwise0} \end{cases}$
POOL	$y_n(x,y) = max(y_n(x,y), y_{n-1}(x+p, y+q))$	$\frac{\partial L}{\partial y_{n-1}}(x+p, y+q) = \begin{cases} 0 & \text{if } y_n(x,y) \neq y_{n-1}(x+p,y+q) \leq 0 \\ \frac{\partial L}{\partial y_n}(x,y) & \text{otherwise0} \end{cases}$

IX. CODE ANALYSIS:

Unlike the previous examples, we will not be making a CNN from scratch with just basic numpy but we will instead use tensorflow abstractions. I will upload my basic CNN with just numpy with complete backpropagation in a later post.

We will be using **input_data.py** to load the data but this time keep in mind that X is of shape [N X28 X 28 X 1]. With this shape, we can apply our filters and convolve on the image. We will also be having our model @ **ckpt_dir** = "CNN_ckpt_dir" and restoring an old model if available.

Once again, we will be splitting our data into batches for processing.

```
import tensorflow as tf
import numpy as np
import input_data
import os

class parameters():

    def __init__(self):
        self.batch_size = 128
        self.num_epochs = 10
        self.ckpt_dir = "CNN_ckpt_dir" # save models here

def load_data():
    # Load the data
    mnist = input_data.read_data_sets("MNIST_data/", one_h
    trainX, trainY, testX, testY = mnist.train.images, mni
                                   mnist.test.images, mnist.test
    trainX = trainX.reshape(-1, 28, 28, 1)
    testX = testX.reshape(-1, 28, 28, 1)
    return trainX, trainY, testX, testY

def generate_batches(batch_size, X, y):

    # Create batches
    num_batches = len(X) // batch_size
    data_X = np.zeros([num_batches, batch_size, 28, 28, 1]
    data_y = np.zeros([num_batches, batch_size, 10], dtype
    for batch_num in range(num_batches):
        data_X[batch_num] = X[batch_num*batch_size:(batch_
        data_y[batch_num] = y[batch_num*batch_size:(batch_
        yield data_X[batch_num], data_y[batch_num]

def generate_epochs(num_epochs, batch_size, X, y):

    for epoch_num in range(num_epochs):
        yield generate_batches(batch_size, X, y)
```

The CNN will have three conv layers with pooling followed by two FC layers. Follow how the shape of the input changes from feeding to output of the third conv layer.

```
def cnn_operations(X, w, w2, w3, w4, w_o,
                   dropout_value_conv, dropout_value_hidden):

    l1a = tf.nn.relu(tf.nn.conv2d(X, w,
        strides=[1,1,1,1], padding='SAME'))
    l1 = tf.nn.max_pool(l1a, ksize=[1,2,2,1],
        strides=[1,2,2,1], padding='SAME')
    l1 = tf.nn.dropout(l1, dropout_value_conv)

    l2a = tf.nn.relu(tf.nn.conv2d(l1, w2,
        strides=[1,1,1,1], padding='SAME'))
    l2 = tf.nn.max_pool(l2a, ksize=[1,2,2,1],
        strides=[1,2,2,1], padding='SAME')
```

```
    l2 = tf.nn.dropout(l2, dropout_value_conv)

    l3a = tf.nn.relu(tf.nn.conv2d(l2, w3,
        strides=[1,1,1,1], padding='SAME'))
    l3 = tf.nn.max_pool(l3a, ksize=[1,2,2,1],
        strides=[1,2,2,1], padding='SAME')
    l3 = tf.reshape(l3,
        [-1, w4.get_shape().as_list()[0]]) # flatten to sh
    l3 = tf.nn.dropout(l3, dropout_value_conv)

    l4 = tf.nn.relu(tf.matmul(l3, w4))
    l4 = tf.nn.dropout(l4, dropout_value_hidden)

    return tf.matmul(l4, w_o)

def init_weights(shape):
    return tf.Variable(tf.random_normal(shape, stddev=0.01

class cnn_model(object):

    def __init__(self):

        # Placeholders
        self.X = tf.placeholder("float", [None, 28, 28, 1]
        self.y = tf.placeholder("float", [None, 10])
        self.dropout_value_conv = tf.placeholder("float")
        self.dropout_value_hidden = tf.placeholder("float"

        # Initalize weights
        w = init_weights([3, 3, 1, 32])       # 3x3x1 conv
        w2 = init_weights([3, 3, 32, 64])     # 3x3x32 con
        w3 = init_weights([3, 3, 64, 128])    # 3x3x64 con
        w4 = init_weights([128 * 4 * 4, 625]) # FC 128 * 4
        w_o = init_weights([625, 10])         # FC 625 inp

        self.logits = cnn_operations(self.X, w, w2, w3, w4
                    self.dropout_value_conv, self.dropout_
        self.cost = tf.reduce_mean(
                    tf.nn.softmax_cross_entropy_with_logit
                                    self.logits, self.y))
        self.optimizer = tf.train.RMSPropOptimizer(0.001,
                                                   self.c

        # Accuracy
        self.correct_prediction = tf.equal(tf.argmax(self.
                                    tf.argmax(self.logits,
        self.accuracy = tf.reduce_mean(tf.cast(
                            self.correct_prediction, tf.fl

        # Components for model saving
        self.global_step = tf.Variable(0, trainable=False)
        self.saver = tf.train.Saver(tf.all_variables())

    def step(self, sess, batch_X, batch_y,
        dropout_value_conv, dropout_value_hidden,
        forward_only=True):

        input_feed = {self.X: batch_X, self.y: batch_y,
                self.dropout_value_conv: dropout_value_con
                self.dropout_value_hidden: dropout_value_h

        if not forward_only:
            output_feed = [self.logits, self.cost,
                           self.accuracy, self.optimizer]
        else:
            output_feed = [self.cost, self.accuracy]

        outputs = sess.run(output_feed, input_feed)

        if not forward_only:
            return outputs[0], outputs[1], outputs[2], out
        else:
            return outputs[0], outputs[1]

def create_model(sess, FLAGS):

    model = cnn_model()

    ckpt = tf.train.get_checkpoint_state(FLAGS.ckpt_dir)
    if ckpt and tf.gfile.Exists(ckpt.model_checkpoint_path
        print("Restoring old model parameters from %s" %
                                ckpt.model_checkpoint_path)
        model.saver.restore(sess, ckpt.model_checkpoint_pa
    else:
        print("Created new model.")
        sess.run(tf.initialize_all_variables())
```

And of course, training is as usual with results for training and validation. Note that we now save our model at the end of each training epoch. When we use **create_model()** next time, we will be starting from the restored point.

```
def train(FLAGS):

    with tf.Session() as sess:

        model = create_model(sess, FLAGS)
        trainX, trainY, testX, testY = load_data()

        # Training
        for epoch_num, epoch in enumerate(
                                generate_epochs(FLAGS.num_
                                                FLAGS.batc
                                                trainX,
                                                trainY)):
            train_cost = []
            train_accuracy = []
            print "Training in progress..."
            for batch_num, (input_X, labels_y) in enumerat
                logits, cost, accuracy, _ = model.step(ses
                                                       inp
```

```
                                                    dro
                                                    dro
                                                    for
                    train_cost.append(cost)
                    train_accuracy.append(accuracy)

                print "Training:"
                print "Epoch: %i, batch: %i, cost: %.3f, accua
                        epoch_num, batch_num,
                        np.mean(train_cost), np.mean(train_acc

                # Validation
                for epoch_num, epoch in enumerate(generate_epo
                                                  num_epochs
                                                  batch_size
                                                  X=testX,
                                                  y=testY)):
                    test_cost = []
                    test_accuracy = []
                    for batch_num, (input_X, labels_y) in enum
                        cost, accuracy = model.step(sess,
                                                    input_X, l
                                                    dropout_va
                                                    dropout_va
                                                    forward_on
                        test_cost.append(cost)
                        test_accuracy.append(accuracy)

                    print "Validation:"
                    print "Epoch: %i, batch: %i, cost: %.3f, a
                        epoch_num, batch_num,
                        np.mean(test_cost), np.mean(test_accur

                # Save checkpoint every epoch.
                if not os.path.isdir(FLAGS.ckpt_dir):
                    os.makedirs(FLAGS.ckpt_dir)
                checkpoint_path = os.path.join(FLAGS.ckpt_dir,
                print "Saving the model."
                model.saver.save(sess, checkpoint_path,
                                 global_step=model.global_step

if __name__ == '__main__':
    FLAGS = parameters()
    train(FLAGS)
```

X. RAW CODE:

GitHub Repo (Updating all repos, will be back up soon!)

Posted in: *Image Recognition*

Tagged: *Tutorials*

← *Vanilla Neural Network*

Pointer Sentinel Mixture Models →

2 THOUGHTS ON "CONVOLUTIONAL NEURAL NETWORKS (CNN)"

Pingback: Image Recognition – The Neural Perspective Edit

Pingback: Convolutional Text Classification – The Neural Perspective Edit

LEAVE A REPLY

Enter your comment here...