Advanced Machine Learning Methods
Project CNN

Thomas Astad Sve thomaasv@stud.ntnu.no
Computer Science, Artificial Intelligence
Norwegian University of Science & Technology
University & Professional Studies ax004206@acsmail.ucsd.edu
University of California, San Diego

Abstract

In this work, I use Convolutional Neural Networks trained on a GPU for classifying images in a tiny imagenet dataset. In particulay I have used the python library lasagne [1] to built on the googlenet [2] architecture.

1. Introduction

I use Convolutional Neural Networks for image classifications on the tiny ImageNet dataset which is very similar to the original ImageNet [3] challenge.

2. Method

2.1 Model

The model used to build the classifier is

2.2 Dataset

3. Experiment

4. Results

Table 1: Shows results for running classification on Googlenet

# classes	# training	# validation	# testing	# epoch	test-loss	Accuracy
200				500	10.9	39.30

5. Conclusion

References

[1] Sander Dieleman, Lasagne is a lightweight library to build and train neural networks in Theano. `https://github.com/Lasagne/Lasagne`

[2]

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. FeiFei. ImageNet: A large-scale hierarchical image database. In CVPR, 2009

Code

main.py

```
import numpy as np
import lasagne
import theano
import theano.tensor as T
import vgg16
import googlenet
import pickle
import time

from preprocess import load_dataset

def iterate_minibatches(inputs, targets, batchsize, shuffle=False):
    assert len(inputs) == len(targets)
    if shuffle:
        indices = np.arange(len(inputs))
        np.random.shuffle(indices)
    for start_idx in range(0, len(inputs) - batchsize + 1, batchsize):
        if shuffle:
            excerpt = indices[start_idx:start_idx + batchsize]
        else:
            excerpt = slice(start_idx, start_idx + batchsize)
        yield inputs[excerpt], targets[excerpt]

def train_network(num_epochs, X_train, y_train, X_val, y_val, train_fn, val_fn):
    # We iterate over epochs:
    for epoch in range(num_epochs):
        # In each epoch, we do a full pass over the training data:
        train_err = 0
        train_batches = 0
        start_time = time.time()
        for batch in iterate_minibatches(X_train, y_train, 500, shuffle=True):
            inputs, targets = batch
            train_err += train_fn(inputs, targets)
            train_batches += 1

        # And a full pass over the validation data:
        val_err = 0
        val_acc = 0
        val_batches = 0
        for batch in iterate_minibatches(X_val, y_val, 500, shuffle=False):
            inputs, targets = batch
            err, acc = val_fn(inputs, targets)
            val_err += err
            val_acc += acc
            val_batches += 1

        # Then we print the results for this epoch:
        print("Epoch {} of {} took {:.3f}s".format(
            epoch + 1, num_epochs, time.time() - start_time))
        print("  training loss:\t\t{:.6f}".format(train_err / train_batches))
        print("  validation loss:\t\t{:.6f}".format(val_err / val_batches))
        print("  validation accuracy:\t\t{:.2f} %".format(
            val_acc / val_batches * 100))


def test_network(X_test, y_test, val_fn):
    test_err = 0
    test_acc = 0
    test_batches = 0
    for batch in iterate_minibatches(X_test, y_test, 500, shuffle=False):
        inputs, targets = batch
        err, acc = val_fn(inputs, targets)
        test_err += err
        test_acc += acc
        test_batches += 1
    print("Final results:")
    print("  test loss:\t\t\t{:.6f}".format(test_err / test_batches))
    print("  test accuracy:\t\t{:.2f} %".format(
        test_acc / test_batches * 100))

def build_parameter_update(network, loss):
    # create parameter update expressions
    params = lasagne.layers.get_all_params(network, trainable=True)
    updates = lasagne.updates.nesterov_momentum(loss, params, learning_rate=0.01,
                                                momentum=0.9)
    return updates


def build_loss(network, target_var):
    # Create a loss expression for training, i.e., a scalar objective we want
    # to minimize (for our multi-class problem, it is the cross-entropy loss):
    prediction = lasagne.layers.get_output(network)
    loss = lasagne.objectives.categorical_crossentropy(prediction, target_var)
    loss = loss.mean()
    return loss

def build_test_loss(network, target_var):
    # Create a loss expression for validation/testing. The crucial difference
    # here is that we do a deterministic forward pass through the network,
    # disabling dropout layers.
    test_prediction = lasagne.layers.get_output(network, deterministic=True)
    test_loss = lasagne.objectives.categorical_crossentropy(test_prediction,
                                                            target_var)
    test_loss = test_loss.mean()
    # As a bonus, also create an expression for the classification accuracy:
    test_acc = T.mean(T.eq(T.argmax(test_prediction, axis=1), target_var),
                      dtype=theano.config.floatX)

    return test_loss, test_acc

def main(num_epochs=500):
    print("Loading data...")
    X_train, y_train, X_val, y_val, X_test, y_test = load_dataset()

    print "X_train: ", X_train.shape, " y_train: ", y_train.shape
    print "X_val: ", X_val.shape, " y_val: ", y_val.shape

    input_var = T.tensor4('inputs')
    target_var = T.ivector('targets')

    print("Building network...")
    #network = vgg16.build_model(input_var)
    network = googlenet.build_model(input_var)
    # Create a loss expression for training
    loss = build_loss(network, target_var)

    # create parameter update expressions
    updates = build_parameter_update(network, loss)

    # Create a loss expression for validation/testing.
    test_loss, test_acc = build_test_loss(network, target_var)

    # Compile a function performing a training step on a mini-batch (by giving
    # the updates dictionary) and returning the corresponding training loss:
    print("Setting training function...")
    train_fn = theano.function([input_var, target_var], loss, updates=updates)

    # Compile a second function computing the validation loss and accuracy:
    print("Setting validation function for loss and accuracy...")
    val_fn = theano.function([input_var, target_var], [test_loss, test_acc])

    # Finally, launch the training loop.
    print("Starting training...")
    train_network(num_epochs, X_train, y_train, X_val, y_val, train_fn, val_fn)

    # After training, we compute and print the test error:
    print("Starting testing...")
    test_network(X_test, y_test, val_fn)

    # Save network
    np.savez('trained_alexnet_200.npz', *lasagne.layers.get_all_param_values(network))

    # And load them again later on like this:
    # with np.load('model.npz') as f:
    #     param_values = [f['arr_%d' % i] for i in range(len(f.files))]
    # lasagne.layers.set_all_param_values(network, param_values)


if __name__ == "__main__":
    main()
```

googlenet.py

```
from lasagne.layers import InputLayer
from lasagne.layers import DenseLayer
from lasagne.layers import ConcatLayer
from lasagne.layers import NonlinearityLayer
from lasagne.layers import GlobalPoolLayer
from lasagne.layers.dnn import Conv2DDNNLayer as ConvLayer
from lasagne.layers.dnn import MaxPool2DDNNLayer as PoolLayerDNN
from lasagne.layers import MaxPool2DLayer as PoolLayer
from lasagne.layers import LocalResponseNormalization2DLayer as LRNLayer
from lasagne.nonlinearities import softmax, linear

def build_inception_module(name, input_layer, nfilters):
    # nfilters: (pool_proj, 1x1, 3x3_reduce, 3x3, 5x5_reduce, 5x5)
    net = {}
    net['pool'] = PoolLayerDNN(input_layer, pool_size=3, stride=1, pad=1)
    net['pool_proj'] = ConvLayer(net['pool'], nfilters[0], 1)
    net['1x1'] = ConvLayer(input_layer, nfilters[1], 1)

    net['3x3_reduce'] = ConvLayer(input_layer, nfilters[2], 1)
    net['3x3'] = ConvLayer(net['3x3_reduce'], nfilters[3], 3, pad=1)

    net['5x5_reduce'] = ConvLayer(input_layer, nfilters[4], 1)
    net['5x5'] = ConvLayer(net['5x5_reduce'], nfilters[5], 5, pad=2)

    net['output'] = ConcatLayer([
        net['1x1'],
        net['3x3'],
        net['5x5'],
        net['pool_proj'],
        ])

    return {'{}/{}'.format(name, k): v for k, v in net.items()}


def build_model(input_var = None):
    net = {}
    net['input'] = InputLayer((None, 3, None, None), input_var = input_var)
    net['conv1/7x7_s2'] = ConvLayer(net['input'], 64, 7, stride=2, pad=3)
    net['pool1/3x3_s2'] = PoolLayer(net['conv1/7x7_s2'],
                                    pool_size=3,
                                    stride=2,
                                    ignore_border=False)
    net['pool1/norm1'] = LRNLayer(net['pool1/3x3_s2'], alpha=0.00002, k=1)
    net['conv2/3x3_reduce'] = ConvLayer(net['pool1/norm1'], 64, 1)
    net['conv2/3x3'] = ConvLayer(net['conv2/3x3_reduce'], 192, 3, pad=1)
    net['conv2/norm2'] = LRNLayer(net['conv2/3x3'], alpha=0.00002, k=1)
    net['pool2/3x3_s2'] = PoolLayer(net['conv2/norm2'], pool_size=3, stride=2)

    net.update(build_inception_module('inception_3a',
                                      net['pool2/3x3_s2'],
                                      [32, 64, 96, 128, 16, 32]))
    net.update(build_inception_module('inception_3b',
                                      net['inception_3a/output'],
                                      [64, 128, 128, 192, 32, 96]))
    net['pool3/3x3_s2'] = PoolLayer(net['inception_3b/output'],
                                    pool_size=3, stride=2)

    net.update(build_inception_module('inception_4a',
                                      net['pool3/3x3_s2'],
                                      [64, 192, 96, 208, 16, 48]))
    net.update(build_inception_module('inception_4b',
                                      net['inception_4a/output'],
                                      [64, 160, 112, 224, 24, 64]))
    net.update(build_inception_module('inception_4c',
                                      net['inception_4b/output'],
                                      [64, 128, 128, 256, 24, 64]))
    net.update(build_inception_module('inception_4d',
                                      net['inception_4c/output'],
                                      [64, 112, 144, 288, 32, 64]))
    net.update(build_inception_module('inception_4e',
                                      net['inception_4d/output'],
                                      [128, 256, 160, 320, 32, 128]))
    net['pool4/3x3_s2'] = PoolLayer(net['inception_4e/output'],
                                    pool_size=3, stride=2)

    net.update(build_inception_module('inception_5a',
                                      net['pool4/3x3_s2'],
                                      [128, 256, 160, 320, 32, 128]))
    net.update(build_inception_module('inception_5b',
                                      net['inception_5a/output'],
                                      [128, 384, 192, 384, 48, 128]))

    net['pool5/7x7_s1'] = GlobalPoolLayer(net['inception_5b/output'])
    net['loss3/classifier'] = DenseLayer(net['pool5/7x7_s1'],
                                         num_units=1000,
                                         nonlinearity=linear)
    net['prob'] = NonlinearityLayer(net['loss3/classifier'],
                                    nonlinearity=softmax)
    return net['prob']
```

preprocess.py

```
import numpy as np
import h5py
from random import shuffle
from math import floor

tiny_imagenet = "http://pages.ucsd.edu/~ztu/courses/tiny-imagenet-200.zip"

def crop_image(image, box):
    # xmin, ymin, xmax, ymax
    resized_image = image[int(box[0]):int(box[2]), int(box[1]):int(box[3])]
    return np.array(resized_image)
```

```
def load_dataset():
    with h5py.File('preprocessed_data/train_set.h5','r') as hf:
        X = hf.get('X')
        X_train = np.array(X, dtype=np.uint8)
        y = hf.get('y')
        y_train = np.array(y, dtype=np.uint8)

    with h5py.File('preprocessed_data/val_set.h5','r') as hf:
        X = hf.get('X')
        X_val = np.array(X, dtype=np.uint8)
        y = hf.get('y')
        y_val = np.array(y, dtype=np.uint8)

    with h5py.File('preprocessed_data/test_set.h5','r') as hf:
        X = hf.get('X')
        X_test = np.array(X, dtype=np.uint8)
        y = hf.get('y')
        y_test = np.array(y, dtype=np.uint8)

    return X_train, y_train, X_val, y_val, X_test, y_test

def save_dataset(filename, X, y = None):
    if y != None:
        with h5py.File("preprocessed_data/" + filename, 'w') as hf:
            hf.create_dataset('X', data=X)
            hf.create_dataset('y', data=y)
    else:
        with h5py.File("preprocessed_data/" + filename, 'w') as hf:
            hf.create_dataset('X', data=X)

def load_training_set(path, Image, wnids):
    import glob, os
    owd = os.getcwd() # Get original path

    images = []
    y = []
    bbox = []
    i = 0

    for class_id in wnids:
        bbox_file = path + class_id + "/" + class_id + "_boxes.txt"
        for line  in open(bbox_file):
            words = line.split()
            #img = Image.open(path + class_id + "/images/" + words[0]).convert('L')
            img = Image.open(path + class_id + "/images/" + words[0])

            image = np.array(img)
            #image_cropped = crop_image(image, words[1:])
            if image.ndim == 3:
                bbox.append(words[1:])
                #image = np.ravel(image)  #Reshape image into columnvector
                image = np.rollaxis(image, 2)
                images.append(image) # Append image to dataset
                y.append(i)
        i = i + 1
        os.chdir(owd) # Reset to original path

    X = np.array(images, dtype=np.uint8)
    y = np.array(y)
    bbox = np.array(bbox)

    return X, y, bbox

def load_val_set(path, Image):
    val_annotations = path + "val_annotations.txt"
    images_path = path + "images/"

    images = []
    y = []
    bbox = []

    for line in open(val_annotations):
        words = line.split()
        image_file = words[0]
        #img = Image.open(images_path + image_file).convert('L') # Read image and convert to grayscale
        img = Image.open(images_path + image_file)
        image = np.array(img)
        #image_cropped = crop_image(image, words[2:])

        #image = np.ravel(image) # Convert the image to a columnvector
        #print image_file, image.shape
        if image.ndim == 3:
            y.append(words[1])
            bbox.append(words[2:])
            image = np.rollaxis(image, 2)
            images.append(image)

    X = np.array(images, dtype=np.uint8)
    y = np.array(y)
    bbox = np.array(bbox)

    return X, y, bbox

def load_test_set(test_path):
    import glob, os
    owd = os.getcwd() # Get original path
    images = []

    for file in glob.glob("*.JPEG"): # For all images in folder
        img = Image.open(file)
        image = np.array(img)
        image = np.rollaxis(image, 2)
        images.append(image)
    os.chdir(owd) # Reset to original path

    return np.array(images)

def split_dataset(X, y, test_size = 0.2, val = False):
    data = zip(X, y)
    shuffle(data)
    X, y = zip(*data)

    if val:
        split_point = int(floor(len(X)*(1 - test_size * 2)))
        X_train, y_train = X[:split_point], y[:split_point]
        X_test_val, y_test_val = X[split_point:], y[split_point:]
        split_point = int(floor(len(X_test_val)*0.5))
        X_test, y_test, X_val, y_val = X_test_val[:split_point], y_test_val[:split_point], \
                                       X_test_val[split_point:], y_test_val[split_point:]

        return np.array(X_train), np.array(y_train), np.array(X_test)\
            , np.array(y_test), np.array(X_val), np.array(y_val)
    else:
        split_point = int(floor(len(X)*(1 - test_size)))
        return X[:split_point], y[:split_point], X[split_point:], y[split_point:]

def generate_dataset(num_classes = 200, save = True):
    import Image
    print("Generating dataset...")
    train_path = "/home/thomas/data/dataset/tiny-imagenet-200/train/"
    val_path = "/home/thomas/data/dataset/tiny-imagenet-200/val/"
    test_path = "/home/thomas/data/dataset/tiny-imagenet-200/test/"
    wnid_file = "/home/thomas/data/dataset/tiny-imagenet-200/wnids.txt"

    wnids = [line.strip() for line in open(wnid_file)]
    wnids = wnids[:num_classes]
    print "Classes: ", len(wnids)
    print "Loading training set"
    X_train, y_train, train_box = load_training_set(train_path, Image, wnids)

    print "Loading validation set"
    if num_classes == 200:
        X, y, train_box = load_training_set(train_path, Image, wnids)
        X_train, y_train, X_test, y_test = split_dataset(X, y, test_size = 0.2)
        X_val, y_val, val_box = load_val_set(val_path, Image)
    else:
        X, y, boxes = load_training_set(train_path, Image, wnids)
        X_train, y_train, X_test, y_test, X_val, y_val = split_dataset(X, y, test_size = 0.15, val = True)
    print "X_val shape: ", X_val.shape, " y_val shape: ", y_val.shape
    print "X_train shape: ", X_train.shape, " y_train shape: ", y_train.shape
    print "X_test shape: ", X_test.shape, " y_test shape: ", y_test.shape

    if save:
        save_dataset("train_set.h5", X_train, y_train)
        save_dataset("val_set.h5", X_val, y_val)
        save_dataset("test_set.h5", X_test, y_test)
        print("Dataset saved")
    else:
        return X_train, y_train, X_val, y_val, X_test, y_test


if __name__ == "__main__":
    generate_dataset(20)
    #X_train, y_train, X_val, y_val, X_test, y_test = load_dataset()
    #print X_train.shape, y_train.shape
    #print X_val, y_val
```