CSCI E-101 (25387):

Introduction to Artificial Intelligence

Companion Course to E-100 Science of Intelligence

© Prof. Brian Subirana

Director, MIT Auto-ID Laboratory

77 Mass Ave, Cambridge MA 02139

Week 4: Coding Assignment

Table of Contents

Data Augmentation and Generalization

You should now be pretty comfortable with the implementational differences between using convolutional and feedforward layers. We will now delve deeper into the differences between these two networks by "augmenting" our data and testing different scenarios. One of the main drawbacks of Deep Learning nowadays is how strongly tied training data is to the performance of a network. Specifically regarding computer vision, the concept of "invariance" is used to refer to images with the same angle of vision, lighting, rotation and location in the field of vision. For example, when we see an object for the first time, even if the lighting changes or if we move around the room, we are still effectively able to tell it apart from the rest. This is not always the case for deep learning models, the next assignment will guide you through different ways to test both your dense and convolutional network, as well as augmenting your data to obtain more training and testing digits on your computer vision task, making your model more robust to variance. Note that you could apply this same train of though for speech recognition or NLP, but for the sake of simplicity we will only look at augmenting the MNIST digit dataset.

The code below is a set of functions that you can use to generate rotated, translated and zoomed MNIST digits for training and testing. You will use these in your next assignment. You do not necessarily need to understand how these work, but rather learn how to use them, which we will discuss next.

```
import matplotlib.pyplot as plt
import numpy as np
from random import random, seed, randint, uniform
import cv2
from scipy.ndimage.interpolation import rotate, zoom

def add_padding(X, padding=10):
    """Add padding to images in array, by changing image size"""
    new_X = []
    for img in X:
        new_image = np.zeros((img.shape[0]+padding*2,img.shape[0]+padding*2))
        new_image[padding:padding+img.shape[0],
                  padding:padding+img.shape[1]] = img
        new_X.append(new_image)
    return np.asarray(new_X)

def move_array(X, transform_range=5):
    """Transform X (image array) with a random move within range."""
    new_X = []
    for img in X:
        moved_x = randint(-transform_range, transform_range)
        moved_y = randint(-transform_range, transform_range)
        translation_matrix = np.float32([[1,0,moved_x], [0,1,moved_y]])
        moved_image = cv2.warpAffine(img, translation_matrix, (img.shape[0],
img.shape[0]))
        new_X.append(moved_image)
```

```
    return np.asarray(new_X)

def rotate_array(X, angle_range=180):
    """Rotate X (image array) with a random angle within angle range."""
    new_X = []
    for img in X:
        angle = random()*angle_range*2-angle_range
        new_image = rotate(img, angle=angle, reshape=False)
        new_X.append(new_image)
    return np.asarray(new_X)

def zoom_array(X, zoom_range_min=0.4, zoom_range_max=1):
    """Zoom X (image array) with a random zoom within zoom range."""
    new_X = []
    for img in X:
        zoom = uniform(zoom_range_min,zoom_range_max)
        zoomed_image = clipped_zoom(img, zoom)
        new_X.append(zoomed_image)

    return np.asarray(new_X)

def clipped_zoom(img, zoom_factor, **kwargs):
    """Clip images to keep image shape while zooming"""
    h, w = img.shape[:2]
    zoom_tuple = (zoom_factor,) * 2 + (1,) * (img.ndim - 2)

    # Zooming out
    if zoom_factor < 1:
        zh = int(np.round(h * zoom_factor))
        zw = int(np.round(w * zoom_factor))
        top = (h - zh) // 2
        left = (w - zw) // 2
        out = np.zeros_like(img)
        out[top:top+zh, left:left+zw] = zoom(img, zoom_tuple, **kwargs)

    # Zooming in
    elif zoom_factor > 1:

        # Bounding box of the zoomed-in region within the input array
        zh = int(np.round(h / zoom_factor))
        zw = int(np.round(w / zoom_factor))
        top = (h - zh) // 2
        left = (w - zw) // 2

        out = zoom(img[top:top+zh, left:left+zw], zoom_tuple, **kwargs)
        trim_top = ((out.shape[0] - h) // 2)
        trim_left = ((out.shape[1] - w) // 2)
        out = out[trim_top:trim_top+h, trim_left:trim_left+w]
```

```
    # If zoom_factor == 1, just return the input array
    else:
        out = img
    return out
```

In order to rotate, zoom or translate a MNIST test dataset, use the functions as follows. Make sure you pad both your training and test sets, this will allow for the whole digit to stay in the image when zooming or translating.

```
X_test = add_padding(X_test)
X_test_rotated = rotate_array(X_test)
X_test_moved = move_array(X_test)
X_test_zoomed = zoom_array(X_test)
```

Remember that you should use these functions BEFORE the datasets are reshaped. You should therefore reshape these the same way you do for either your dense or convolutional layers. For example, to use the rotated dataset for your dense neural network, you would use this.

```
num_pixels = X_test[0].shape[0]* X_test[0].shape[1]
X_test_rotated = X_test_rotated.reshape(X_test.shape[0], num_pixels).astype('float32')
```

MNIST Data Augmentation and Invariance Assignment

This assignment is based on the code explained in the "*Deep Learning with Keras*" section, make sure you review it before you get on with this assignment.

All the work you do in this section should be properly recorded and compiled into a final report, combined with the other exercises. You will find the guidelines for this report at the end of the assignment.

- You are provided with functions that help you pad, rotate, translate and zoom in or out your MNIST data. You can find examples on how to use this code above. Evaluate your convolutional and dense layer networks with a rotated test dataset, a zoomed test dataset and a translated test dataset. Record performances and discuss. *Remember to pad both your training and test datasets!*
- Create a larger training dataset for your networks by randomly rotating, zooming and translating samples. Train both your convolutional and dense networks (note that this will take longer as you will have a lot more data than in the previous assignments).
- Once again, evaluate your convolutional and dense layer networks with the same previous rotated, zoomed and translated test datasets. Record performances and discuss.