EEE 498 ML with application to FPGAs

All homework is handed in online. Don't hand in pictures of the code files, hand in the actual code files so I can see if they work. Hand in images of outputs of your code (displays, plots …) converted to pdf. If there are questions to be answered answer them in PowerPoint, Word, or on paper and convert to pdf (It doesn't actually matter how you get to the pdf).

Organize your work so that the answer to each problem can be easily identified. You will not get credit if your submission is poorly organized, or if the grader cannot find the answers.

You can use any language python, c, C++, MATLAB,…, to do coding though Python is likely the easiest and easiest to translate from lectures.

Homework 6 Your own little CNN

In this assignment we will try to exercise some of the steps of a very simple CNN to understand the details. The objective isn't to write a CNN as much as understand the steps in more detail. In part 1 we will do forward propagation, in part 2 backward propagation.

In this assignment 7 by 7 black and white image arrays are available with numbers 1, 2 and 3 in them written in slightly different styles. The objective is to build a CNN algorithm and train it to recognize them but that is in part 2. In part 1 we will build the forward propagation and the convolution/correlation functions.

The CNN will be very simple with 2 layers illustrating the power of this method. The first layer is a 2D convolution with a stride of 2 and valid padding which should yield a 3X3 matrix and the second is a very simple ANN with a 1X3 weight vector which when matrix multiplied by this 3X3 will result in a weighted sum of features giving classification 1, 2, and 3 which are represented in one hot coded vector target below.

The method will be to standardize, subtract of the mean and divide by the standard deviation for each image. Then a 2D convolution is done with a weight vector which will be trained to detect 1, 2, and 3. For instance 1 looks like this (training and test sets given below).

```
image7by7 = np.zeros([9,Image_dim,Image_dim],float)
image7by7[0,1,:] = np.array([0,0,0,1,0,0,0])
image7by7[0,2,:] = np.array([0,0,1,1,0,0,0])
image7by7[0,3,:] = np.array([0,0,0,1,0,0,0])
image7by7[0,4,:] = np.array([0,0,0,1,0,0,0])
image7by7[0,5,:] = np.array([0,0,1,1,1,0,0])
```

The convolution is done using a 2D convolution function you write. The prediction based on weights can be determined by a forward propagation algorithm. The image is convolved with weight array W1 and then that is convolved with weight array W2 designed to classify in 3 resulting classes 1, 2, or 3. The cost function is determined by the sum of squared errors between the target and prediction for each image. Each set of weights is determined by moving in the direction of the negative gradient of the cost function with respect to that weight.

Be careful copying code from office products to Spyder or other editors, some characters are changed like single quotes. Sometimes invisible characters occur causing you to get an error on a line until you completely retype it.

1) Write your own convolution Y=f*g and correlation Y=f•g algorithms. Use a 7X7 matrix for f, with all zeros except the center value, which should be 1. For g use a 3X3 matrix with values [[1,2,3],[3,4,5],[6,7,8]]. Write a 2D convolution to convolve the 7X7 and 3X3. Use valid padding. Correlate and the convolve f with g and show the results. Submit your code, your results and answer to this question. (stride = 1 for this problem)

A square reference correlation function and convolution function are shown below for reference so that you can check your work.

```
import scipy.signal as ss
def conv2Dd_(image,W,stride,Conv):
  if (Conv):
    y = ss.convolve2d(image, W, mode='valid')  ## valid padding
  else:
    y = ss.correlate2d(image, W, mode='valid') ## valid padding
  Xdim = len(image[0])//stride
  x = np.zeros([Xdim,Xdim],float)
  if stride>1:                                  ## implement stride
    for i in range(0,Xdim):
      for j in range(0,Xdim):
        x[i,j] = y[i*stride,j*stride]
  else:
    x = y
  return(x)
```

This is a simple, square matrix scipy 2d convolution that **you can use to check your algorithm**
note: that this algorithm may have some weird scaling (1/sqrt(2pi) would have made sense, but it isn't quite that), a scalar multiplied by the result, but that shouldn't matter to the result, the scaling should be consistent over the elements of the result

In fact, a convolution is a correlation with a 180 degree rotated weight matrix. To avoid doing this rotation and to make the results more straight forward the correlation is often used instead of convolution. If you rotate the weight matrix and use correlation you are effectively doing a convolution which is what we are doing here.

Discrete convolution

$$(f * g)_n = \sum_{m=-\infty}^{m=\infty} f_{n-m}\, g_m$$

Remember these are 1D formulas

correlation

$$(f \cdot g)_n = \sum_{m=-\infty}^{m=\infty} f_{n+m}\, g_m$$

So, they could be used in 2D only by flattening the portion of the matrix overlapping the weights, and the weights on each step

2) In this problem you will write the forward propagation for a CNN followed by a simple ANN which will take its outputs and turn them into classifications. Using the 2D convolution algorithm you wrote above do forward propagation, or prediction based on these given weights. (stride = 2 for this problem)

Try the convolution weights

```
W1 = np. array([[ 1.6975548,  -0.07326141, -0.41880725],
                [ 0.12228276, -0.19572004,  0.81986898],
                [ 0.8876136,  -1.8629187,  -0.97661273]])
##  And the ANN weights
W2 = np. array([ 1.10485759,  0.2120758,  -1.31693339])
```

Don't forget to standardize each image (shown below) so that the mean is 0 and the standard deviation is 1. If you don't do this the weights I have given you may not work.

To get X3 matrix multiply the output of the correlation/convolution function W2. Use a quantizer which sets the maximum prediction to 1 and the others to 0 for each image. Maybe like this:

```
##
##  categorize
##
def categorize(X):
   XX = np.zeros(len(X),float)
   maxind = X.tolist().index(max(X.tolist()))
   XX[maxind] = 1.0
   return(XX)
```

Predict each of the "images" and compare to the targets. The algorithm should classify your training and test cases correctly, but you may have to decide whether you should rotate the weights or not.

Turn in your code, your results, organized so that the grader can find all your code data and results for each problem. If they can't find something they will assume 0.

Be careful copying code from office products to Spyder or other editors, some characters are changed like single quotes. Sometimes invisible characters occur causing you to get an error on a line until you completely retype it.

```
import numpy as np
import matplotlib.pyplot as plt
Image_dim = 7
stride = 2
Ydim = 3
Wdim = 3
##
##  target
##
target = np.array([[1,0,0],[1,0,0],[1,0,0],[0,1,0],[0,1,0],[0,1,0],[0,0,1],[0,0,1],[0,0,1]],float)
##
##  training data
##
## 0
##  this is a form that is easier to work with, the other form works too
image7by7 = np.zeros([9,Image_dim,Image_dim],float)
```

```
image7by7[0,1,:] = np.array([0,0,0,1,0,0,0])
image7by7[0,2,:] = np.array([0,0,1,1,0,0,0])
image7by7[0,3,:] = np.array([0,0,0,1,0,0,0])
image7by7[0,4,:] = np.array([0,0,0,1,0,0,0])
image7by7[0,5,:] = np.array([0,0,1,1,1,0,0])
## 1
image7by7[1,1,:] = np.array([0,0,1,1,0,0,0])
image7by7[1,2,:] = np.array([0,0,0,1,0,0,0])
image7by7[1,3,:] = np.array([0,0,0,1,0,0,0])
image7by7[1,4,:] = np.array([0,0,0,1,0,0,0])
image7by7[1,5,:] = np.array([0,0,0,1,0,0,0])
## 2
image7by7[2,1,:] = np.array([0,0,0,1,0,0,0])
image7by7[2,2,:] = np.array([0,0,0,1,0,0,0])
image7by7[2,3,:] = np.array([0,0,0,1,0,0,0])
image7by7[2,4,:] = np.array([0,0,0,1,0,0,0])
image7by7[2,5,:] = np.array([0,1,1,1,1,1,0])
## 3
image7by7[3,1,:] = np.array([0,1,1,1,1,1,0])
image7by7[3,2,:] = np.array([0,1,0,0,1,0,0])
image7by7[3,3,:] = np.array([0,0,0,1,0,0,0])
image7by7[3,4,:] = np.array([0,1,1,0,0,0,0])
image7by7[3,5,:] = np.array([0,1,1,1,1,1,0])
## 4
image7by7[4,1,:] = np.array([0,0,1,1,1,0,0])
image7by7[4,2,:] = np.array([0,0,0,0,1,0,0])
image7by7[4,3,:] = np.array([0,0,0,1,0,0,0])
image7by7[4,4,:] = np.array([0,0,1,0,0,0,0])
image7by7[4,5,:] = np.array([0,0,1,1,1,0,0])
## 5
image7by7[5,1,:] = np.array([0,0,1,1,0,0,0])
image7by7[5,2,:] = np.array([0,1,0,0,1,0,0])
image7by7[5,3,:] = np.array([0,0,0,1,0,0,0])
image7by7[5,4,:] = np.array([0,0,1,0,0,0,0])
image7by7[5,5,:] = np.array([0,1,1,1,1,1,0])
## 6
image7by7[6,1,:] = np.array([0,1,1,1,1,1,0])
image7by7[6,2,:] = np.array([0,1,0,0,0,1,0])
image7by7[6,3,:] = np.array([0,0,1,1,1,1,0])
image7by7[6,4,:] = np.array([0,0,0,0,0,1,0])
image7by7[6,5,:] = np.array([0,1,1,1,1,1,0])
## 7
image7by7[7,1,:] = np.array([0,0,1,1,1,0,0])
image7by7[7,2,:] = np.array([0,1,0,0,0,1,0])
image7by7[7,3,:] = np.array([0,0,0,1,1,0,0])
image7by7[7,4,:] = np.array([0,1,0,0,0,1,0])
image7by7[7,5,:] = np.array([0,0,1,1,1,0,0])
## 8
image7by7[8,1,:] = np.array([0,1,1,1,1,1,0])
```

```
image7by7[8,2,:] = np.array([0,1,0,0,0,1,0])
image7by7[8,3,:] = np.array([0,0,0,1,1,1,0])
image7by7[8,4,:] = np.array([0,1,0,0,0,1,0])
image7by7[8,5,:] = np.array([0,1,1,1,1,1,0])
## print('image 1 \n', image7by7)
##
##  test data
##
targett = np.array([[1,0,0],[0,0,1],[0,1,0]],float)
##
image7by7t = np.zeros([3,7,7],float)
image7by7t[0,1,:] = np.array([0,0,0,1,0,0,0])
image7by7t[0,2,:] = np.array([0,0,0,1,0,0,0])
image7by7t[0,3,:] = np.array([0,0,0,1,0,0,0])
image7by7t[0,4,:] = np.array([0,0,0,1,0,0,0])
image7by7t[0,5,:] = np.array([0,0,0,1,1,0,0])
## print('image 1 \n', image7by7t0)
image7by7t[1,1,:] = np.array([0,0,1,1,1,0,0])
image7by7t[1,2,:] = np.array([0,1,0,0,0,1,0])
image7by7t[1,3,:] = np.array([0,0,0,1,1,1,0])
image7by7t[1,4,:] = np.array([0,1,0,0,0,1,0])
image7by7t[1,5,:] = np.array([0,0,1,1,1,0,0])
## print('image 1 \n', image7by7t1)
## test image a 2
image7by7t[2,1,:] = np.array([0,0,1,1,1,1,0])
image7by7t[2,2,:] = np.array([0,1,0,0,1,0,0])
image7by7t[2,3,:] = np.array([0,0,0,1,0,0,0])
image7by7t[2,4,:] = np.array([0,0,1,0,0,0,0])
image7by7t[2,5,:] = np.array([0,1,1,1,1,1,0])
```

Also one way to 180 degree rotate a matrix

```
##
##  reverse order columns and then rows
##
def M180deg(M):
   return(np.flip(np.flip(M,axis=1),axis=0))
```

3) Copy this algorithm to a different file and **add back propagation** to determine the weights to classify these images into class 1, 2 and 3. Start with random weights, I used these

```
##
##  training
##  initial settings
##
np.random.seed(1)
W1 = (np.random.rand(Wdim,Wdim)-0.5)
print(W1)
W2 = (np.random.rand(Wdim)-0.5)
```

print(W2)

(stride = 2 for this problem, must be consistent with problem 2, right?)

You will need to write a function to find dzdW1 and a function to find dzdW2. Show your algorithms clearly. You can check your result using autograd but write your own function.

You should get the weights shown in 2, but your weights may be scaled differently.

Turn in the code and the results of your run. Make which weights you found clear. Apply the weights you found to the training data and the test dataset and show how well you did in matching each.

Hand in all output as a screen grab for each problem converted to pdf. It is a good practice to paste well organized and labeled images into PowerPoint, answer all questions in the PowerPoint file, and then convert the PowerPoint file to pdf. Hand in all your code files. If code files are missing you will not get credit. It is typically easier to grade individual files rather than a zipped file, but either is accepted.