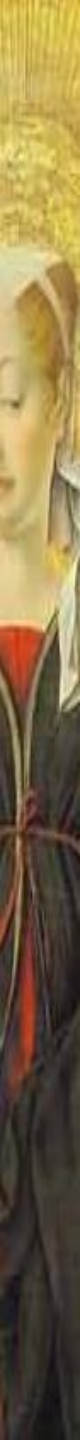
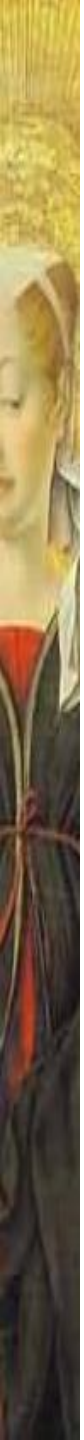


Neural Networks

CS212 Topics in Computing 2
Module 2
Lecture 4

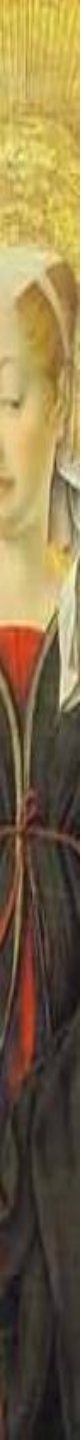
Spring 2018

Dr. Dmitri Roussinov
dmitri.roussinov@strath.ac.uk



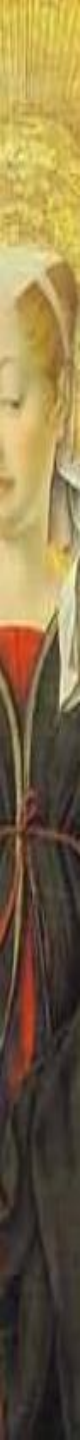
Lab 2

- No mass glitches
- Everybody survived

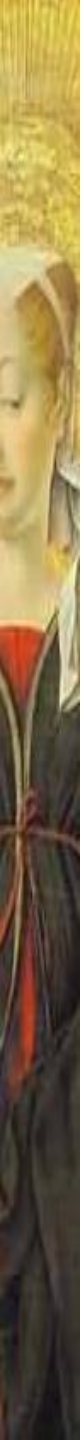


Big picture so far

- Learned: derivatives and gradients are crucial to train the models by optimizing their accuracy (reducing errors)
- Learned: how vector dot products can help to express the model in simpler and more efficient way
- Learned: how we can use Theano to efficiently obtains the gradients
- Just started: how we can use that to create an Artificial Neural Network so solve real life applications
  - E.g. *handwritten digits recognition*



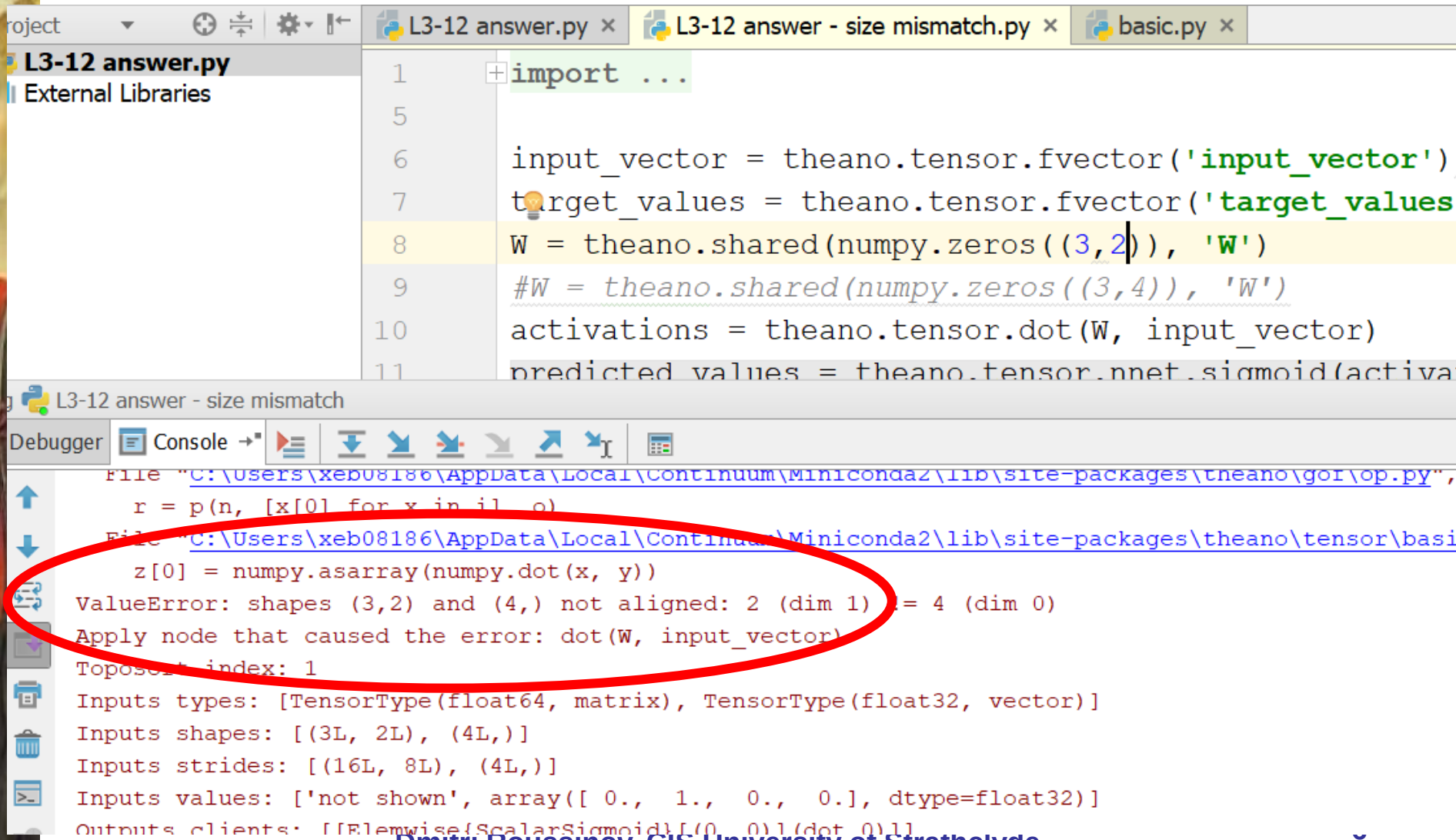
Dot Products Review



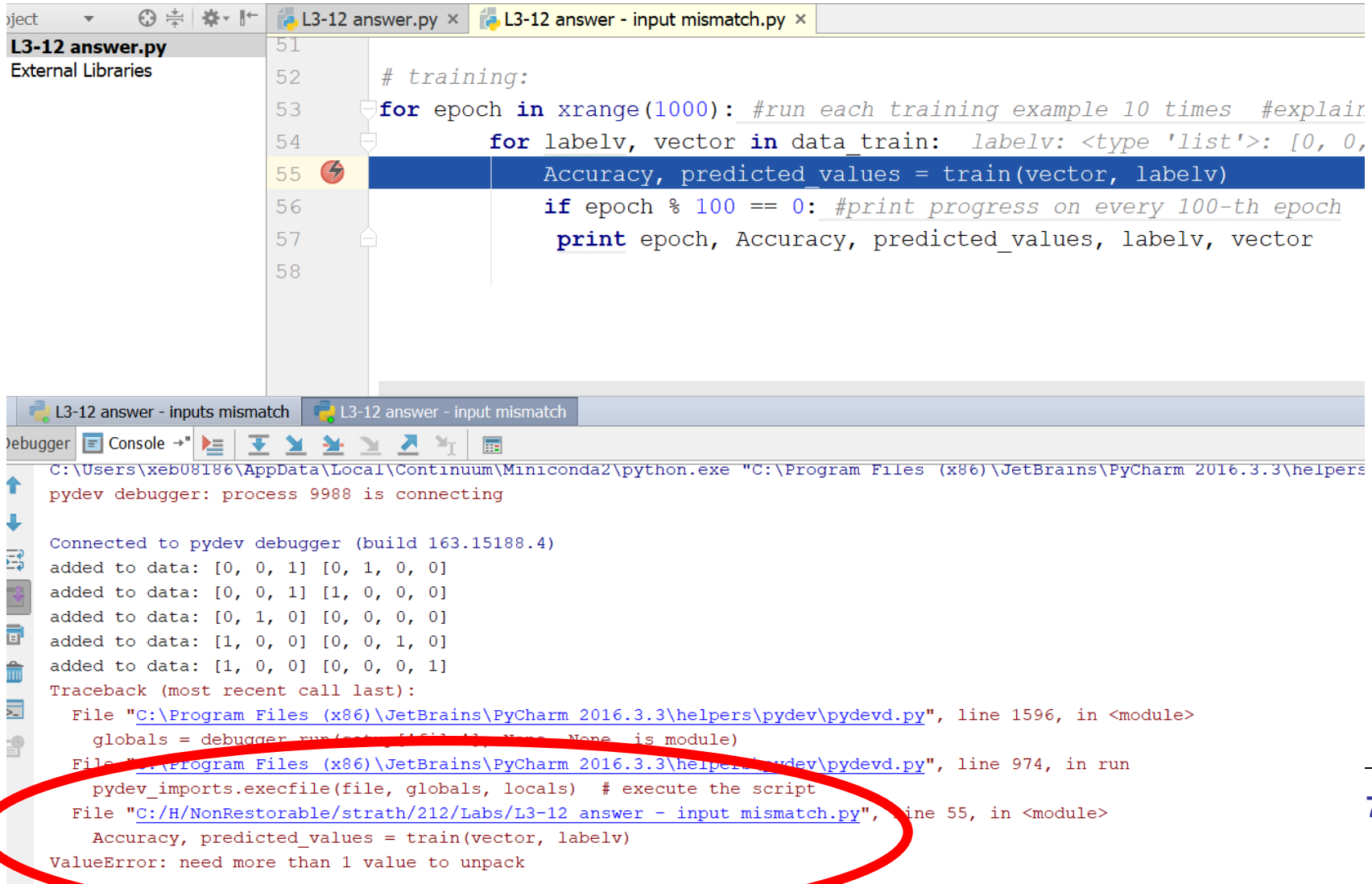
Lab 3 Tips

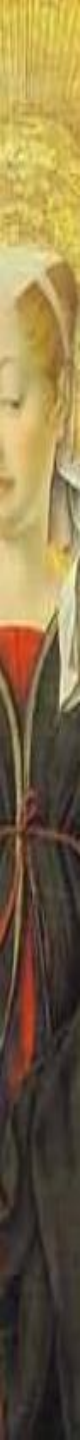
- Tasks 10-11 are explained in Lecture 3 slides
- Task 12 expects you to read from a file
  - Template posted
  - Asks to use different variable sizes

Possible Error: Size Mismatch



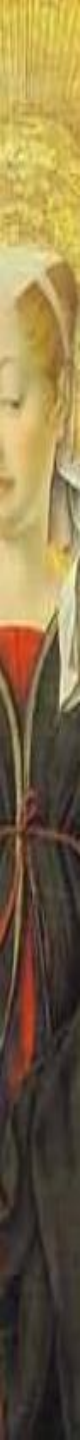
Possible Error: Input/Output Mismatch



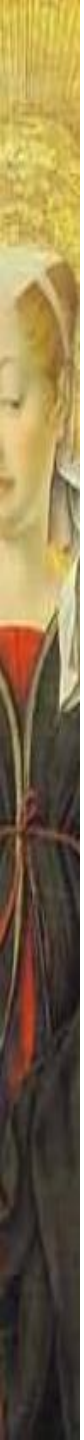


Lab 3-14: Last Part

- Using a Neural Network to implement handwritten digits recognition
- Separately, due to snow cancellations 2 weeks back
- Due before our test time
- Extra sessions this week (I will post times)
  - You can get help to wrap up
  - You can get it marked
- Right after the test you can get Lab 3 (all parts) marked but
  - We may not be able to give you enough help
  - We may be not be able to mark all – so some will be marked base on your uploads
  - So do you best to show at extra sessions to get more feedback and help!

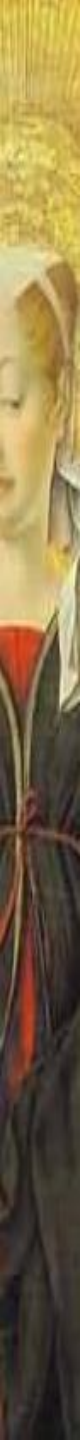


Lab 3-14: Last Part Competition Format



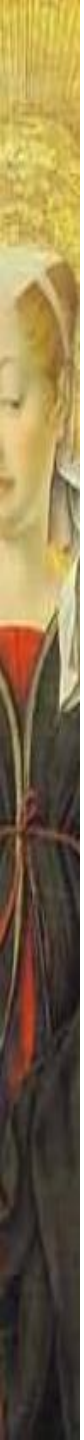
Let’s try again a few demos

- vqa.cloudcv.org
- http://alteredqualia.com/xg/examples/morphs_paintings.html



Plan for Today

- A few tips on Lab 3
- More about Neural Networks
  - Application to recognizing MNIST digits
- A few tips on last part Lab 3-14
- Possibly: questions about the sample test
- Wrapping up!

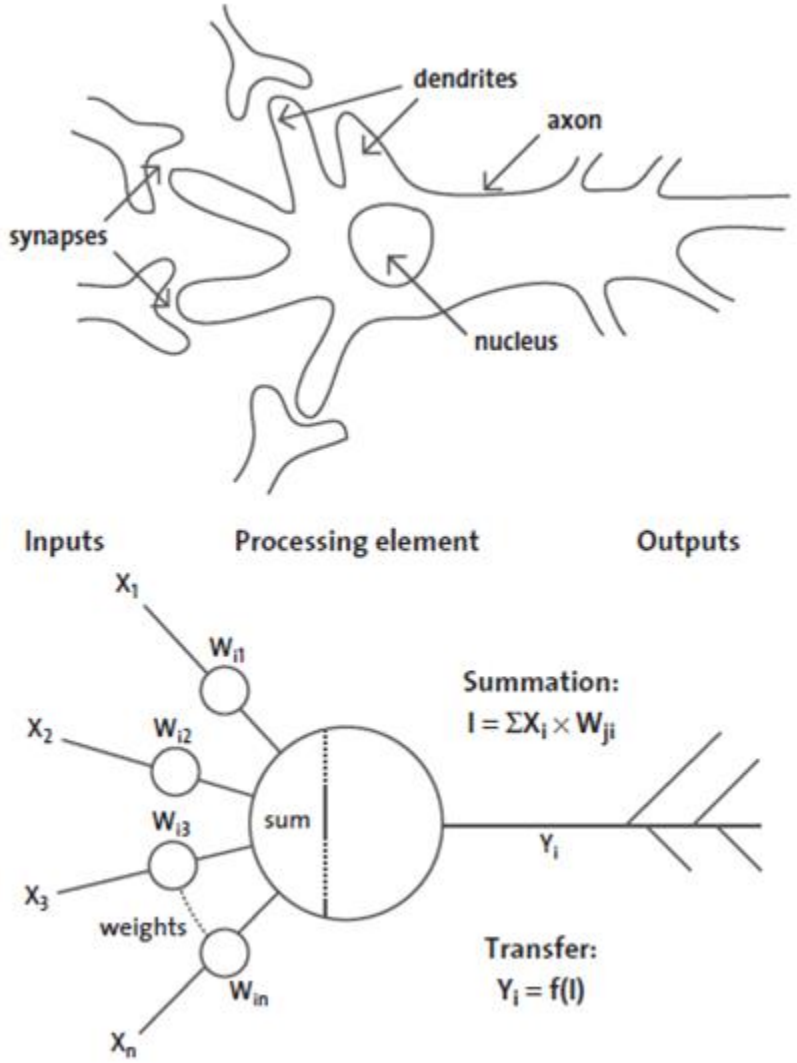


High Level Steps when Optimizing (Training) using Theano

- Define your prediction model
  - e.g. linear *y = w1*x1+w2*x2*
- Define your accuracy
  - e.g. negative sum of square errors
- Obtain the gradient
  - e.g. *G=theano.tensor.grad(y, [w1,w2])*
- Define updates for the training function
  - e.g. *[(w1, w1+.1*G[0]), (w2, w2+.1*G[1])]*
- Call the training function in the loop

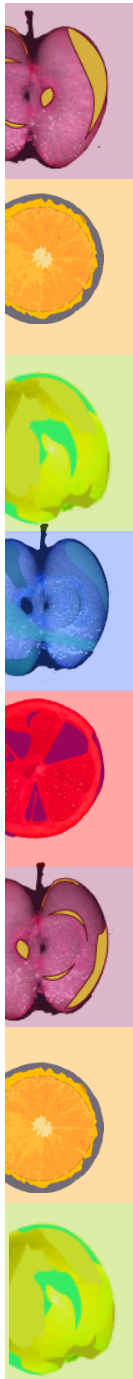
- Done!

Artificial Neural Network: Biological Motivation

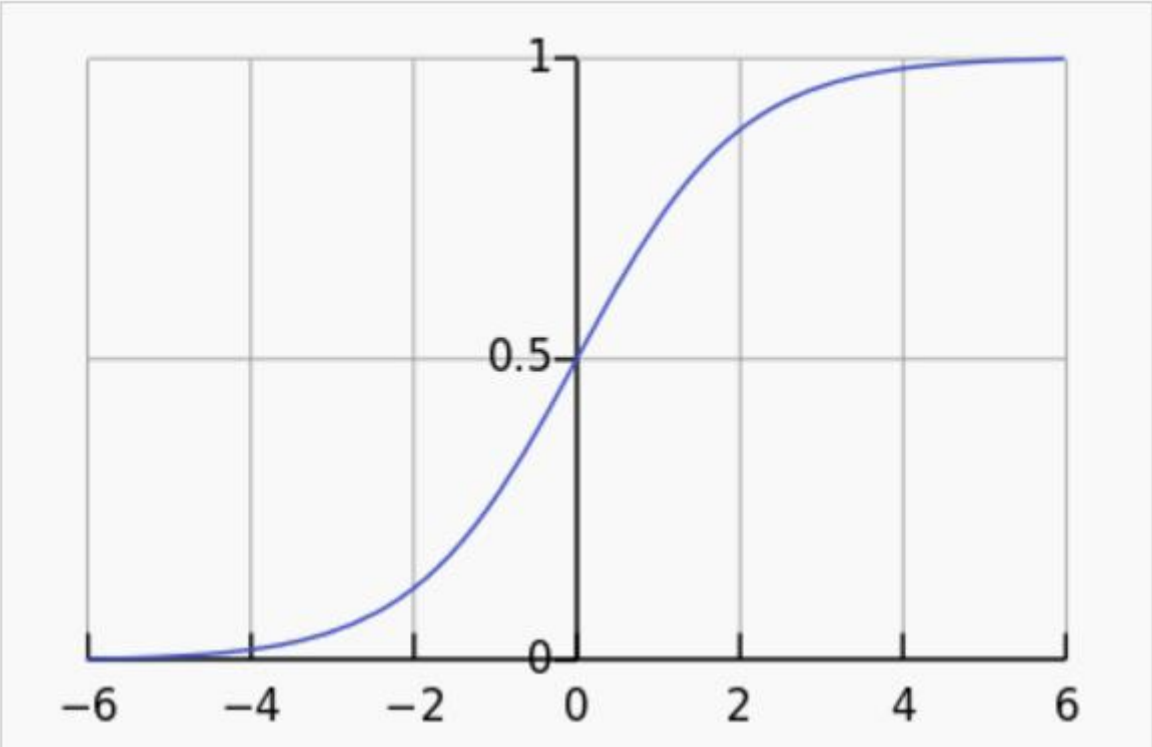


Things To Note

- Input variables are converted into output variables
- One node takes several inputs and produces one output
- Linear conversion with weights can be used in the simplest case
- Outputs can be used as inputs for other nodes
- Thus, multiple layers can be implemented
- Function *f* provides non-linearity
- Most often it is sigmoid



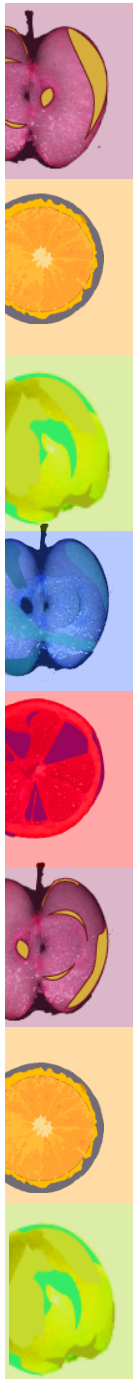
Non linearity: sigmoid function



$$S(t) = \frac{1}{1 + e^{-t}}.$$

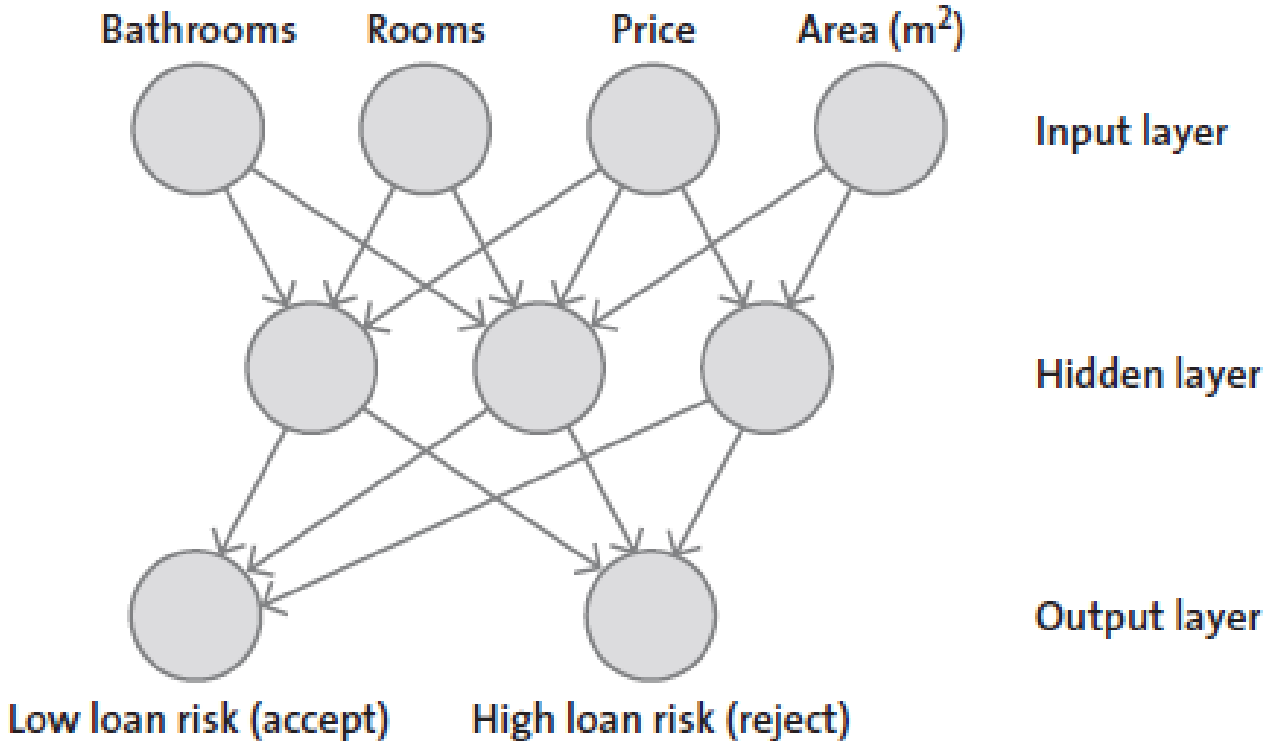
Things To Note

- First, the value is calculated by adding all inputs with weights
- Then, the sigmoid function is applied to the value:
  - For large positive values sigmoid approaches 1
  - For large negative values sigmoid approaches 0
  - For intermediate values, the transition from 0 to 1 is a smooth mathematical function
  - That allows a number of simplifications when obtaining its gradient



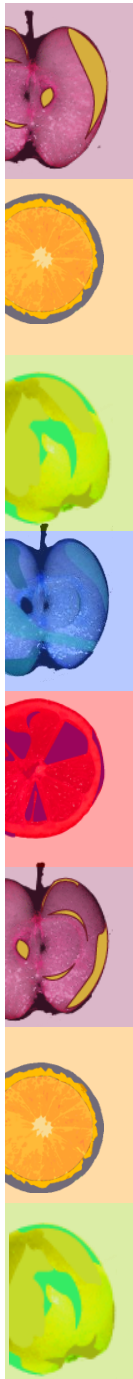
ANN: Architecture

Figure 5.13 The architecture of a artificial neural network for a hypothetical predictor of domestic property loan risk



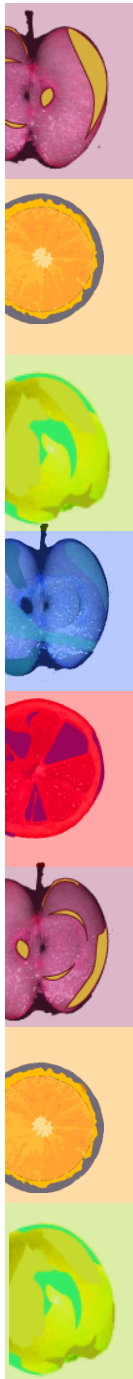
Terminology

- Each node is called a neuron
- Neurons are organized into layers
- Normally, neurons are connected to other neurons only on adjacent layers
- Input layer is the one that represents the features (AKA independent variables)
  - E.g. #bathrooms, price, area
- Output layer is the one that represents the predictions (AKA label, AKA dependent variable)
  - E.g. accept/reject loan decision
- Hidden is a layer that is neither input nor output



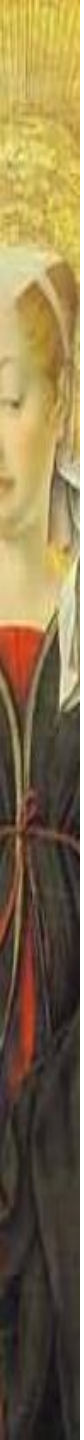
Backpropagation

- **Gradient Ascent/Descent** is a common method of training artificial neural networks so as to minimize the prediction errors
- When applied to Neural Networks, it is called **Backpropagation**
- We already know how use it in general case and define the model in **Theano**
- **That saves out a lot of hassle**



MNIST task

- What are the challenges?



Overall Template Posted: 2 Screens of code

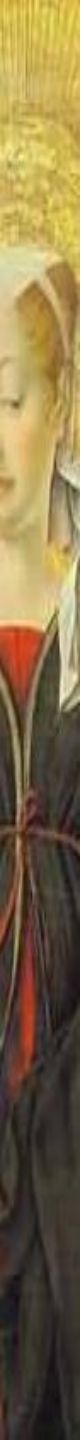
- Templates and Test/Exam:
  - The details from the next two slides are not directly tested
  - The program code from the template posted for lab 3 is not directly tested
  - All other slides are testable unless excluded

Screen 1: Dummy Theano model and reading data

```
import ...
input_vector = theano.tensor.fvector('input_vector') #theano variable representing image
target_values = theano.tensor.fvector('target_values') #theano variable representing the l
#This is just a template: it does not learn anything, and always returns the digit "0":
params = []
updates = []
theano.config.on_unused_input='ignore'
predicted_class = theano.shared(0, 'predicted_class')
Accuracy = theano.shared(0, 'Accuracy')
#Change this to something meaningful and it will work!

# defining Theano functions for training and testing the model:
train = theano.function([input_vector, target_values], [Accuracy, predicted_class], update
test = theano.function([input_vector, target_values], [Accuracy, predicted_class], allow_i
    #'allow_input_downcast=True' is needed to avoid any issues converting between 64 and 3
def read_dataset(path): #The function that reads training or testing data and returns it a
    number_of_images = len(open(path).readlines()) / 28 #28 lines per each image
    f = open(path)
    dataset = [] #starts with an empty container
    for i in range(number_of_images):
     data_vector = [] #we start with empty data vector, and then we read the data for each
     for l in range(28): #each image takes 28 lines
```

Screen 2: training/testing

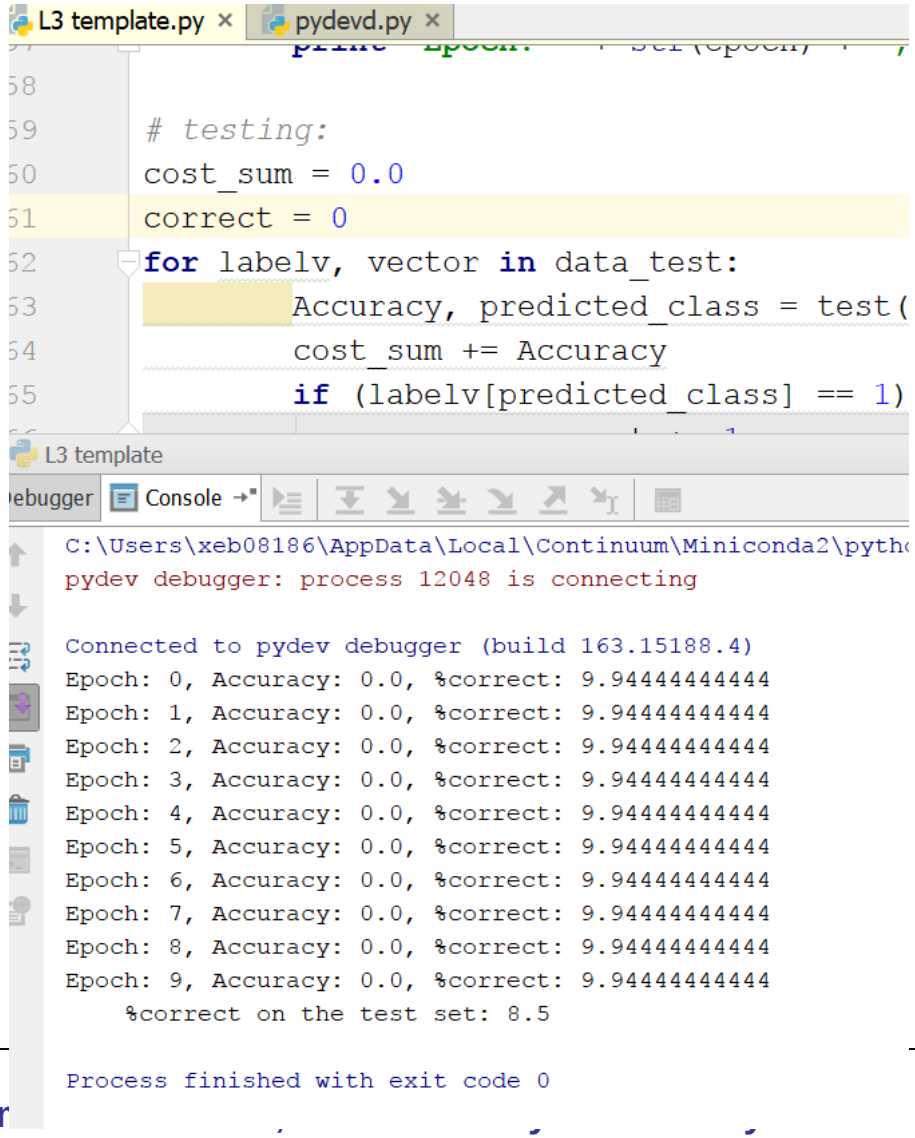
```
#reading the data:
data_test = read_dataset("test.txt")
data_train = read_dataset("train.txt")

# training
for epoch in xrange(10):
        cost_sum = 0.0
        correct = 0
        for labelv, vector in data_train:
            Accuracy, predicted_class = train(vector, labelv)
            cost_sum += Accuracy
            if (labelv[predicted_class] == 1):
                correct += 1
        print "Epoch: " + str(epoch) + ", Accuracy: " + str(cost_sum) + ", %correct: " + str(flo

# testing:
cost_sum = 0.0
correct = 0
for labelv, vector in data_test:
        Accuracy, predicted_class = test(vector, labelv)
        cost_sum += Accuracy
        if (labelv[predicted_class] == 1):
                    correct += 1
print "\t%correct on the test set: " + str(float(correct) / len(data_test)*100)
```

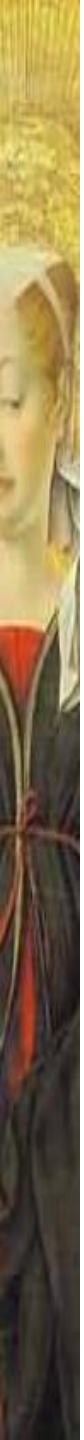


Things to Note

- Outline similar to the one for Task 12 (Sentiment from data in a file)
  - Added Theano function 'test' to test the trained accuracy
    - It is almost same as 'train'
    - It calculate the predictions but does not obtain gradients and does not change W
- 'read_dataset' function similar to Task 12, but hints:
  - Input format a bit different (explained below)
  - Sizes are now different

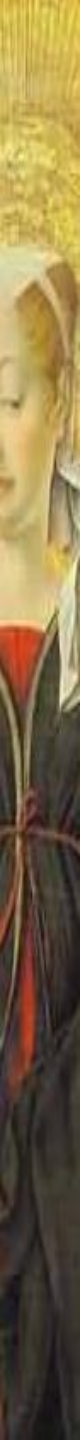
Output: Nothing Useful Yet





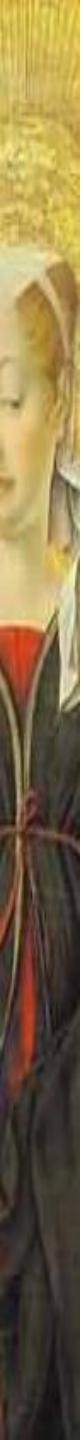
More to Notice:

- One epoch finishes when each training example is fed once
- We print accuracy (%correct) after each epoch
- After 10 epochs, the overall accuracy (%correct) is computed over a dataset that the model has not seen called a test-set
- This is a standard procedure to simulate real-life scenario
- Currently it is 10% since our model always tells it is the digit “0”



Data Format

```
7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
7   0   0   0   0   0   0   84  185 159 151 60  36  0   0   0   0   0   0   0   0   0   0   0
7   0   0   0   0   0   0   222 254 254 254 254 241 198 198 198 198 198 198 198 198 170 52  0
7   0   0   0   0   0   0   67  114 72  114 163 227 254 225 254 254 254 250 229 254 254 140 0
7   0   0   0   0   0   0   0   0   0   0   0   17  66  14  67  67  67  59  21  236 254 106 0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   83  253 209 18  0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   22  233 255 83  0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   129 254 238 44  0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   59  249 254 62  0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   133 254 187 5   0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   9   205 248 58  0   0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   126 254 182 0   0   0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   75  251 240 57  0   0   0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   19  221 254 166 0   0   0   0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   3   203 254 219 35  0   0   0   0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   38  254 254 77  0   0   0   0   0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   31  224 254 115 1   0   0   0   0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   133 254 254 52  0   0   0   0   0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   61  242 254 254 52  0   0   0   0   0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   121 254 254 219 40  0   0   0   0   0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   121 254 207 18  0   0   0   0   0   0   0   0
7   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   116 125 171 255 255 150 93  0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   169 253 253 253 253 253 253 218 30  0   0   0   0
2   0   0   0   0   0   0   0   0   0   169 253 253 253 213 142 176 253 253 122 0   0   0   0   0
2   0   0   0   0   0   0   0   0   52  250 253 210 32  12  0   6   206 253 140 0   0   0   0   0
2   0   0   0   0   0   0   0   0   77  251 210 25  0   0   0   122 248 253 65  0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   31  18  0   0   0   0   209 253 253 65  0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   117 247 253 198 10  0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   0   76  247 253 231 63  0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   0   128 253 253 144 0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   176 246 253 159 12  0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   25  234 253 233 35  0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   198 253 253 141 0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   78  248 253 189 12  0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   19  200 253 253 141 0   0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   134 253 253 173 12  0   0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   248 253 253 25  0   0   0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   248 253 253 43  20  20  20  20  5   0   5   20  20  37  150
2   0   0   0   0   0   0   0   0   0   248 253 253 253 253 253 253 253 168 143 166 253 253 253 253
2   0   0   0   0   0   0   0   0   0   174 253 253 253 253 253 253 253 253 253 253 253 249 247 247
2   0   0   0   0   0   0   0   0   0   0   118 123 123 123 166 253 253 253 155 123 123 41  0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
2   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

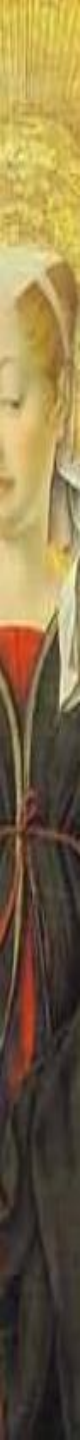


Things To Note:

- 28 lines per image
- Label (digit) is the very first number on each line
- 9000 images to train
- 1000 images to test

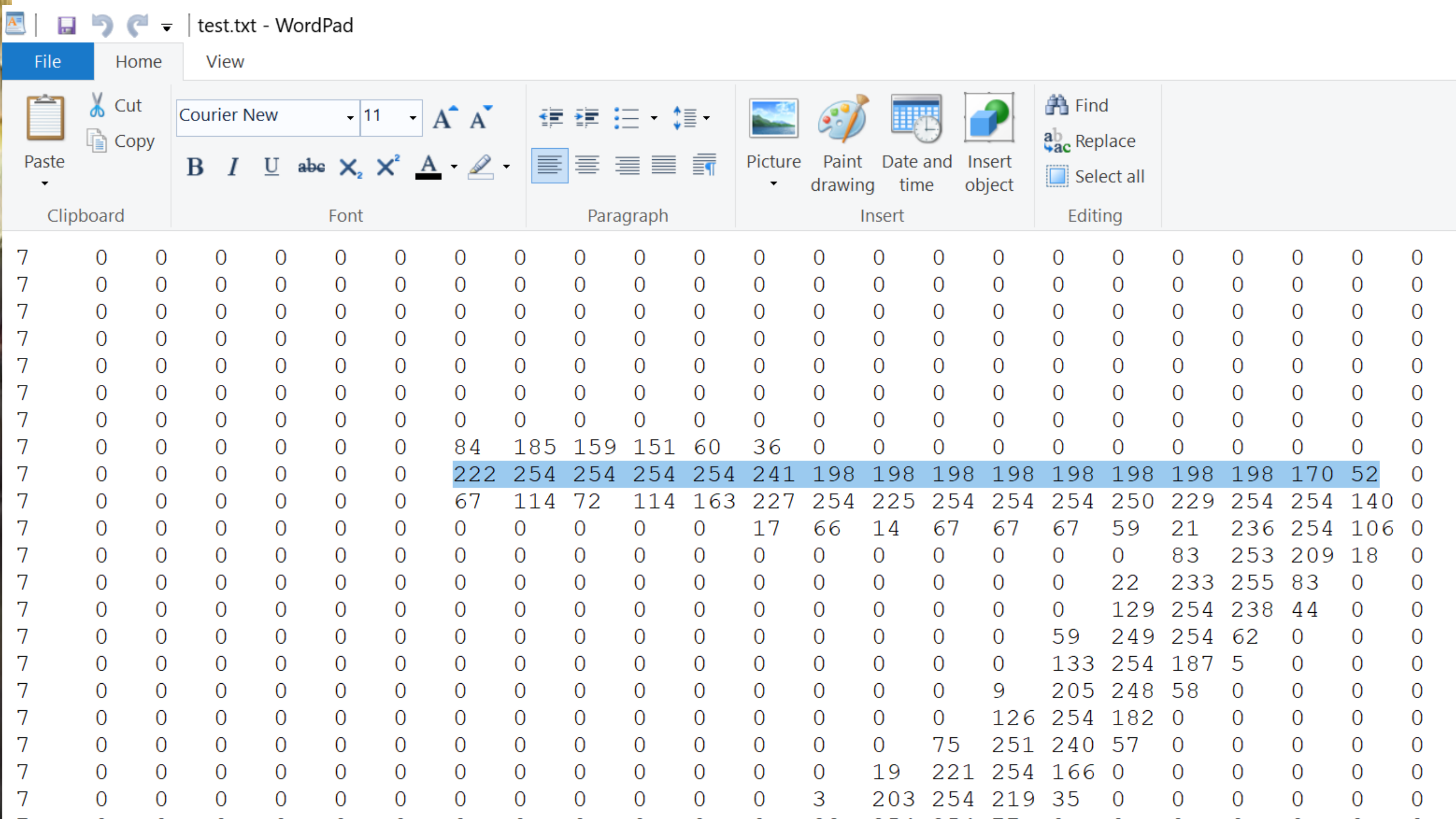
The code to create one-hot vectors in the template

```
def read_dataset(path): #The function that reads training or testing data
    number_of_images = len(open(path).readlines()) / 28 #28 lines per eac
    f = open(path)
    dataset = [] #starts with an empty container
    for i in range(number_of_images):
     data_vector = [] #we start with empty data vector, and then we read
     for l in range(28): #each image takes 28 lines
            line = f.readline()
            line_parts = line.split() #split the line into all the number
            assert(len(line_parts) == 29) #should be total of 29: the lab
            label = int(line_parts[0]) #very first number in the file is
            assert (0 <= label <= 9) #only digits 0-9 are allowed as labe
            #now we will create "one-hot vector":
            label_vector = [0,0,0,0,0,0,0,0,0,0] #a vector of 10 zeroes
            label_vector[label] = 1 #except 1 for the label
            data_vector += [float(line_parts[i])/255 for i in xrange(1,
            #we divide by 255 so the pixel brightness is represented by a

     dataset.append((label_vector, data_vector))
    return dataset
```



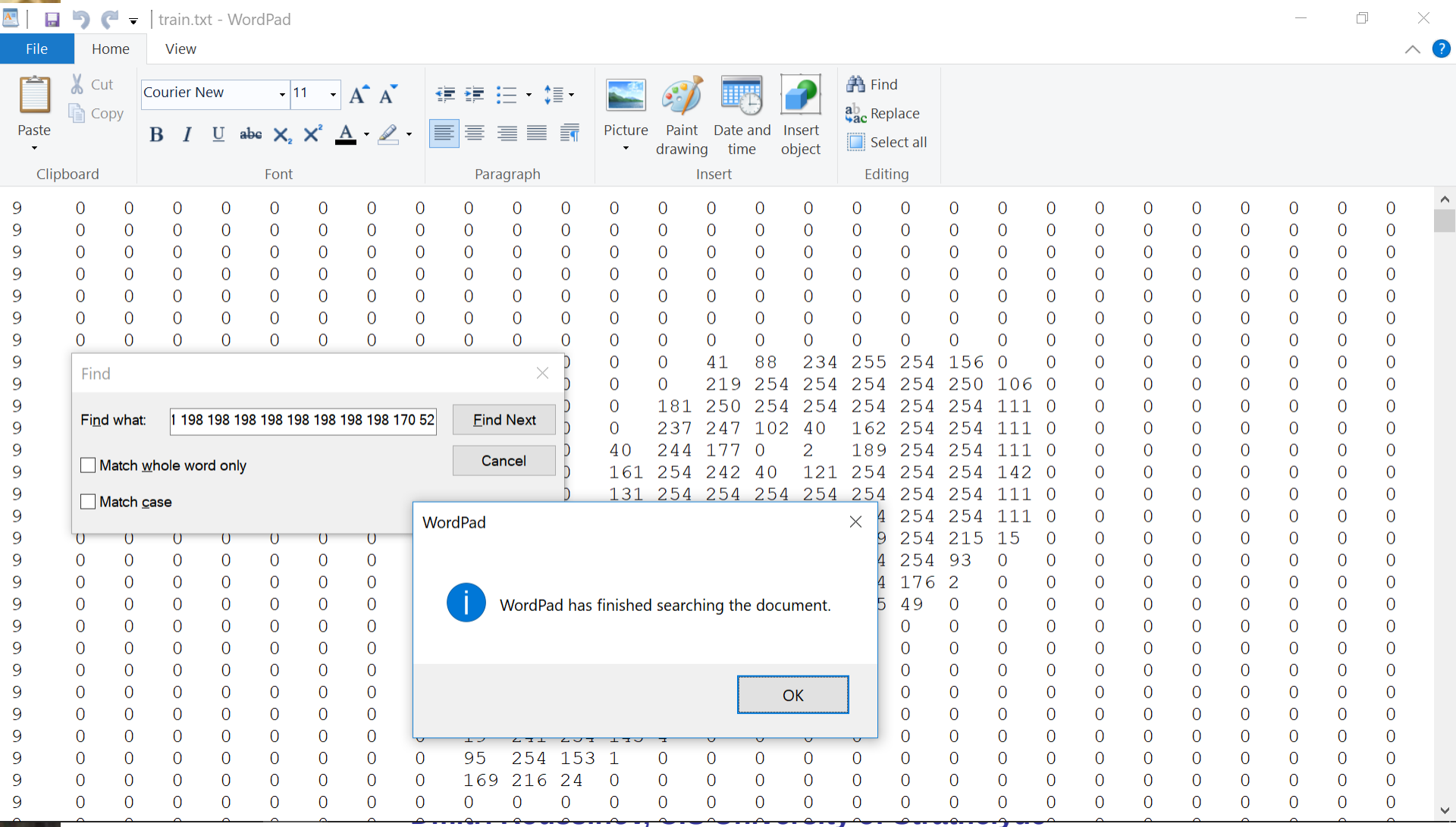
How to verify train/test don’t overlap (Task 14-a)

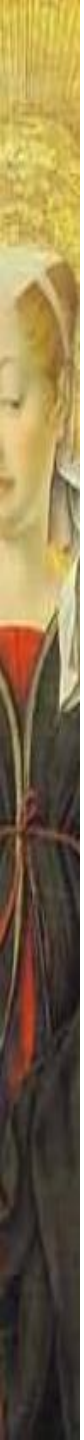
Select some “non trivial” line in test



7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	84	185	159	151	60	36	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	222	254	254	254	254	241	198	198	198	198	198	198	198	198	170	52	0
7	0	0	0	0	0	0	67	114	72	114	163	227	254	225	254	254	254	250	229	254	254	140	0
7	0	0	0	0	0	0	0	0	0	0	0	17	66	14	67	67	67	59	21	236	254	106	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	83	253	209	18	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	22	233	255	83	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	129	254	238	44	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	59	249	254	62	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	133	254	187	5	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	205	248	58	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	126	254	182	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	75	251	240	57	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	19	221	254	166	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	3	203	254	219	35	0	0	0	0	0	0

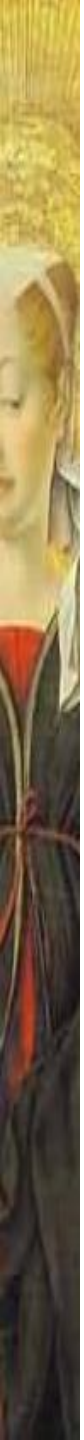
Search for it in train





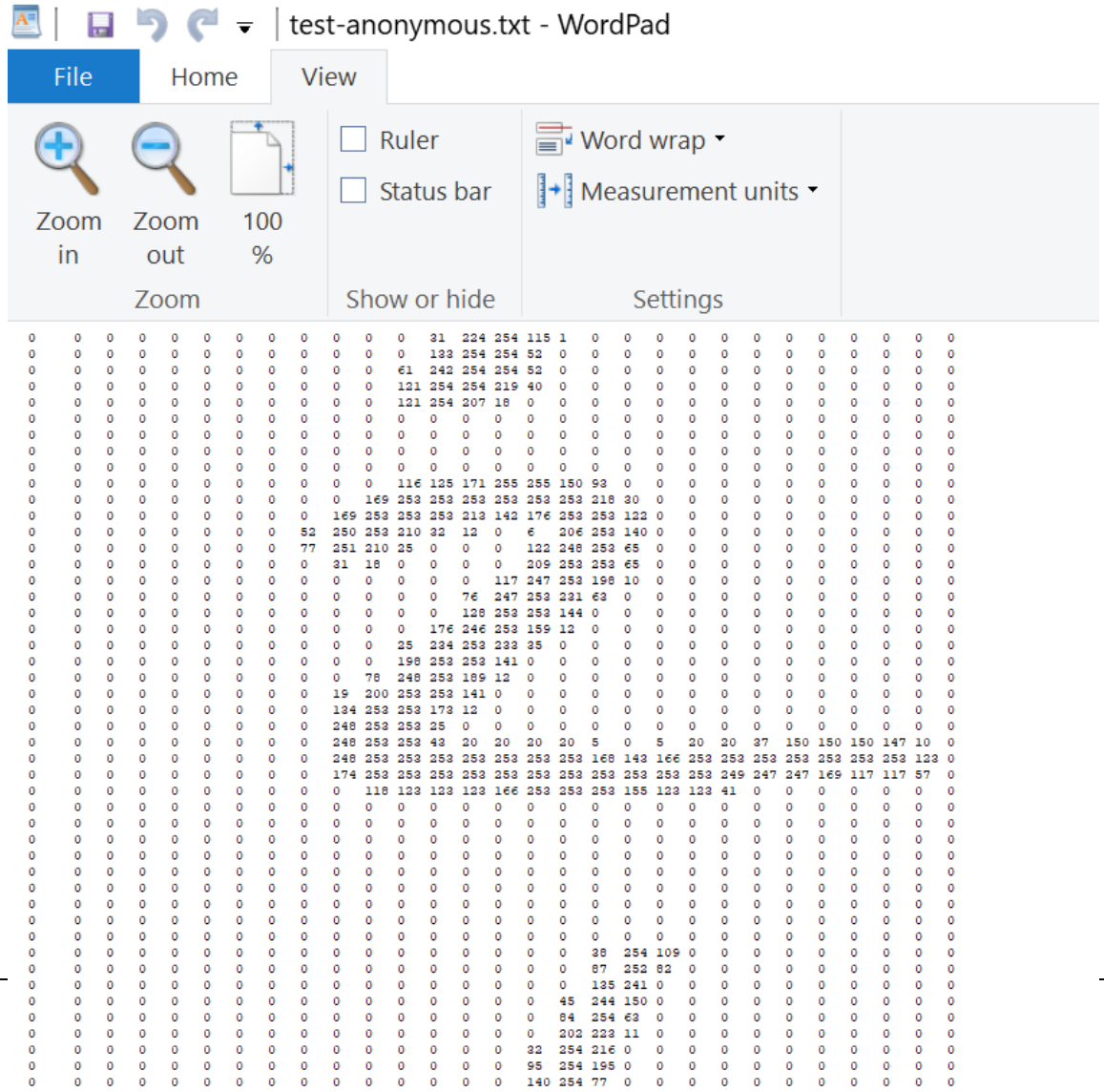
What to check

- If there is not such line, we are fine
- Can check a few more randomly selected sequences
- If we find it, verify that the entire image is still not the same

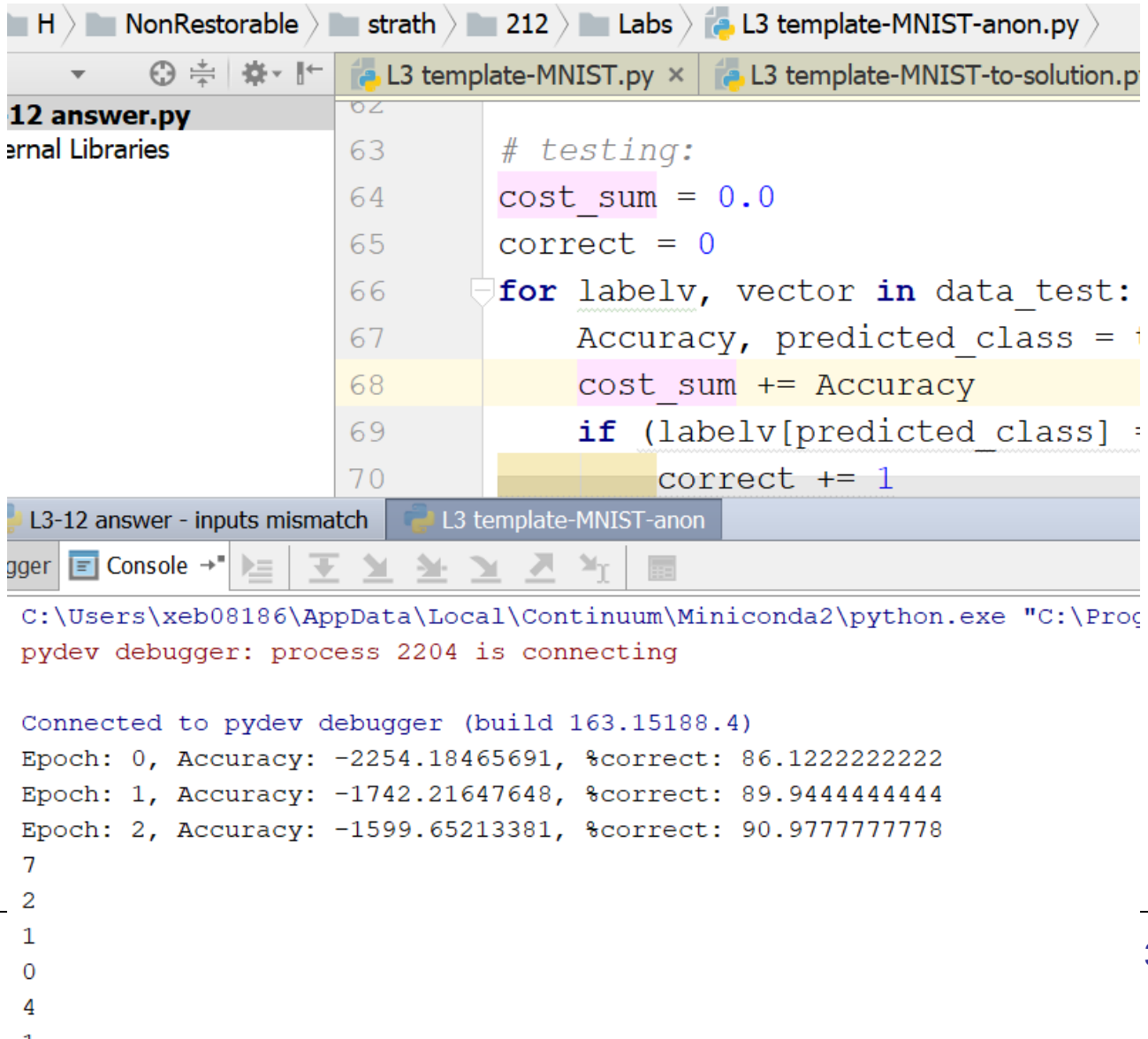


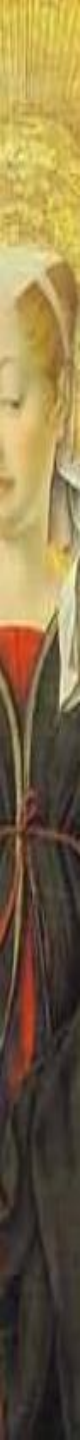
Let’s do altogether

Testing without giving the labels



Still Works!


H
NonRestorable
strath
212
Labs
L3 template-MNIST-anon.py
L3 template-MNIST.py
L3 template-MNIST-to-solution.p
12 answer.py
ernal Libraries
63 # testing:
64 cost_sum = 0.0
65 correct = 0
66 for labelv, vector in data_test:
67 Accuracy, predicted_class =
68 cost_sum += Accuracy
69 if (labelv[predicted_class]
70 correct += 1
L3-12 answer - inputs mismatch
L3 template-MNIST-anon
gger
Console
C:\Users\xeb08186\AppData\Local\Continuum\Miniconda2\python.exe "C:\Prog
pydev debugger: process 2204 is connecting
Connected to pydev debugger (build 163.15188.4)
Epoch: 0, Accuracy: -2254.18465691, %correct: 86.1222222222
Epoch: 1, Accuracy: -1742.21647648, %correct: 89.9444444444
Epoch: 2, Accuracy: -1599.65213381, %correct: 90.9777777778
7
2
1
0
4

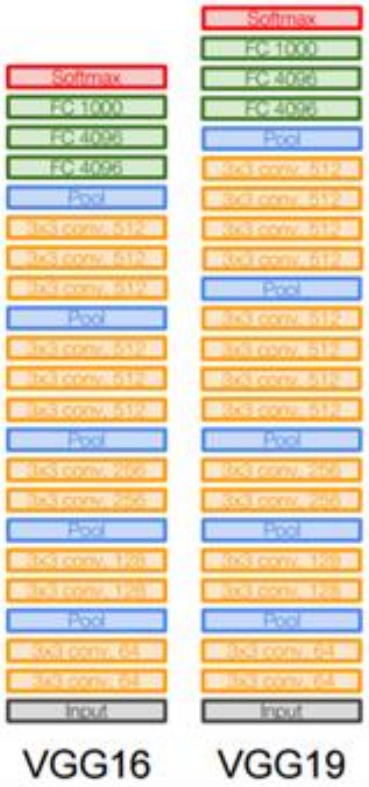


Let’s try to create more layers

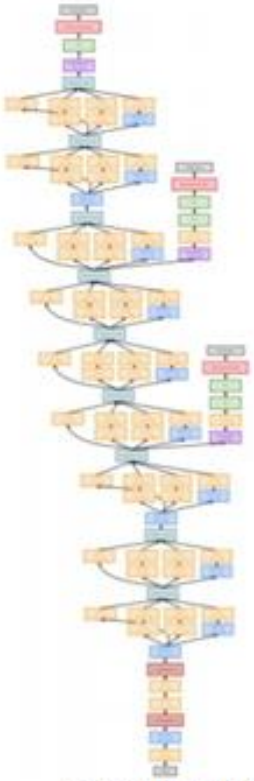
- And do a question from the sample

Modern Computer Vision Architecture

Last Time: CNN Architectures



VGG16 VGG19



GoogLeNet

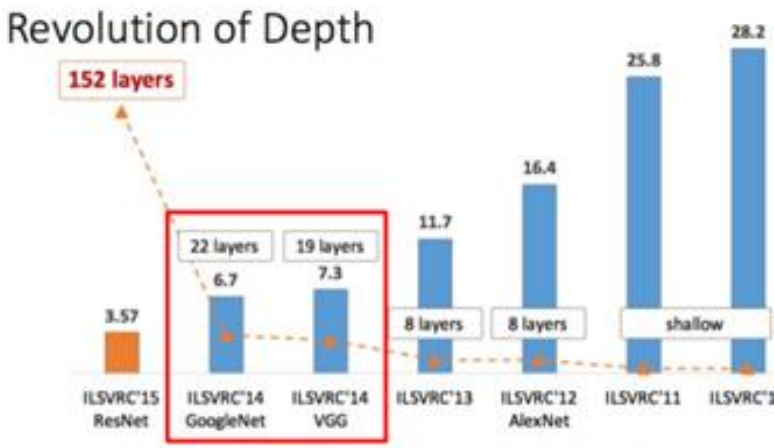
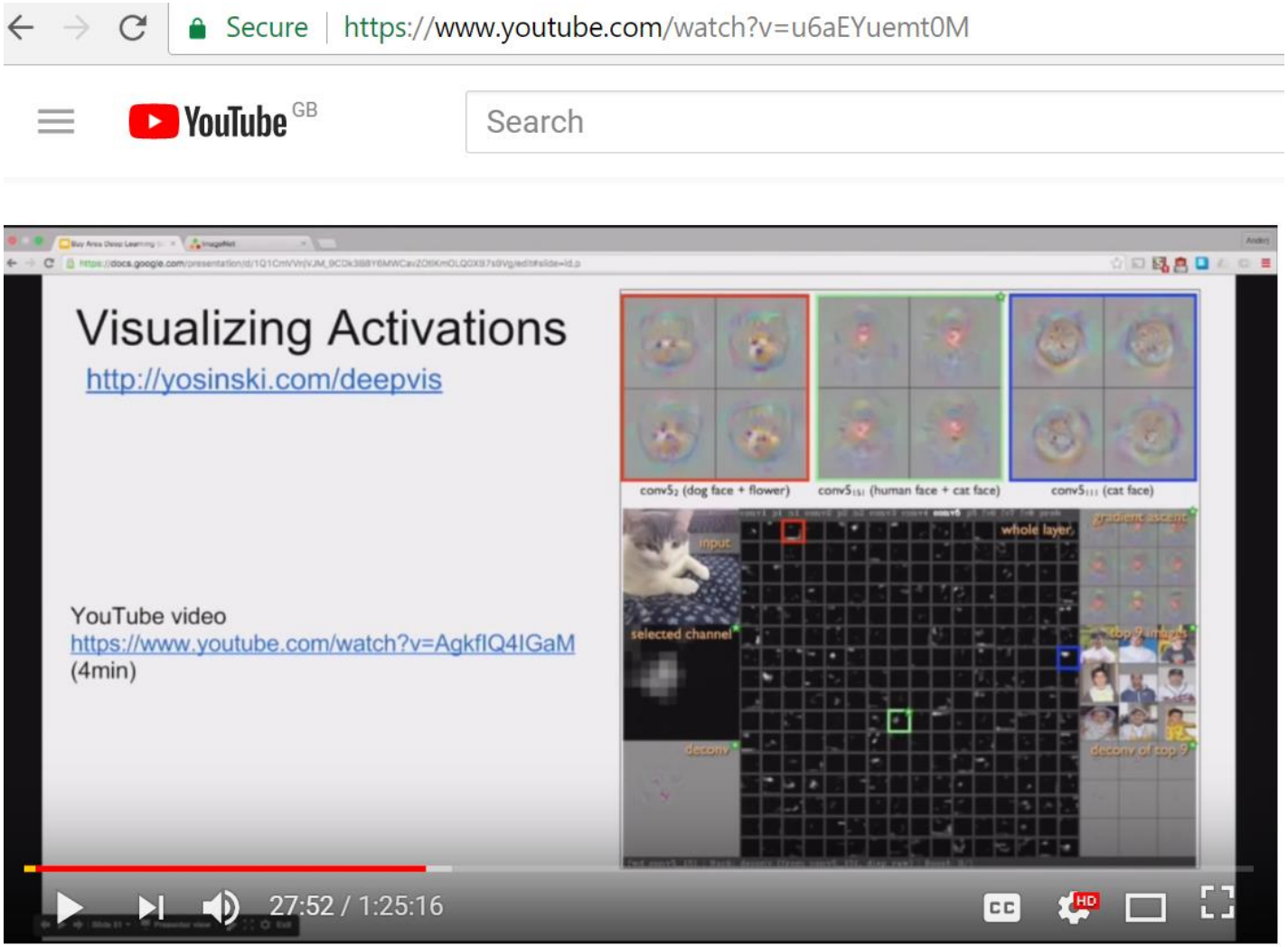


Figure copyright Kaiming He, 2016. Reproduced with permission.

- For illustration only: Not to be tested

How this works in larger networks



Deep Learning for Computer Vision (Andrej Karpathy, OpenAI)

- (for illustration only)