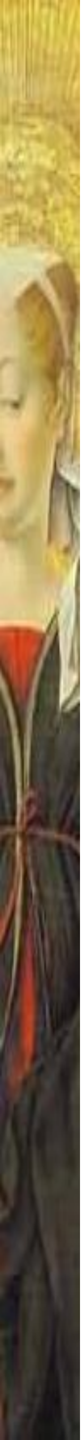
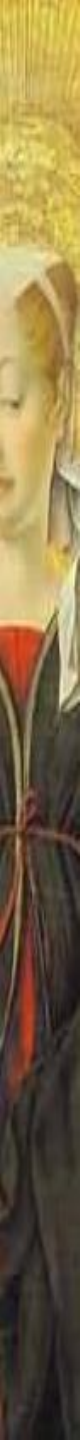


Neural Networks

CS212 Topics in Computing 2
Module 2
Lecture 3

Spring 2018

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



Lab 1

- No mass glitches
- Everybody survived

Debugging

- Like solving a murder mystery …

Error May be in The Middle …

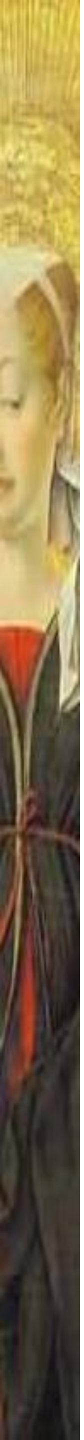
ug sentiment for slides

Debugger Console

```
    output = f()
  File "C:\Users\xeb08186\AppData\Local\Continuum\Miniconda2\lib\site-packages\theano\compile\function_module.py", line 898, ir
    storage_map=getattr(self.fn, 'storage_map', None))
  File "C:\Users\xeb08186\AppData\Local\Continuum\Miniconda2\lib\site-packages\theano\gof\link.py", line 325, in raise_with_op
    reraise(exc_type, exc_value, exc_trace)
  File "C:\Users\xeb08186\AppData\Local\Continuum\Miniconda2\lib\site-packages\theano\compile\function_module.py", line 884, ir
    self.fn() if output_subset is None else\
  File "C:\Users\xeb08186\AppData\Local\Continuum\Miniconda2\lib\site-packages\theano\gof\op.py", line 872, in rval
    r = p(n, [x[0] for x in i], o)
  File "C:\Users\xeb08186\AppData\Local\Continuum\Miniconda2\lib\site-packages\theano\tensor\basic.py", line 5785, in perform
    z[0] = numpy.asarray(numpy.dot(x, y))
ValueError: shapes (5,2) and (3,) not aligned: 2 (dim 1) != 3 (dim 0)
Apply node that caused the error: dot(TensorConstant{[[1 1]
 [2..1]
 [0 1]]}, W)
Toposort index: 0
Inputs types: [TensorType(int32, matrix), TensorType(float64, vector)]
Inputs shapes: [(5L, 2L), (3L,)]
Inputs strides: [(8L, 4L), (8L,)]
Inputs values: ['not shown', array([ 0.,  0.,  0.])]
Outputs clients: [[Elemwise{Composite{(i0 * (i1 - i2))}}[(0, 1)](TensorConstant{(1L,) of -2.0}, dot.0, TensorConstant{[-2  2  0

Backtrace when the node is created(use Theano flag traceback.limit=N to make it longer):
  File "C:\Program Files (x86)\JetBrains\PyCharm 2016.3.3\helpers\pydev\pydevd.py", line 1596, in <module>
    globals = debugger.run(setup['file'], None, None, is_module)
  File "C:\Program Files (x86)\JetBrains\PyCharm 2016.3.3\helpers\pydev\pydevd.py", line 974, in run
    pydev_imports.execfile(file, globals, locals)  # execute the script
  File "C:/H/NonRestorable/strath/104/Labs/Lab2/sentiment for slides.py", line 8, in <module>
    predictions = theano.tensor.dot(texts, W) #this is our linear model

HINT: Use the Theano flag 'exception_verbosity=high' for a debugprint and storage map footprint of this apply node.
```

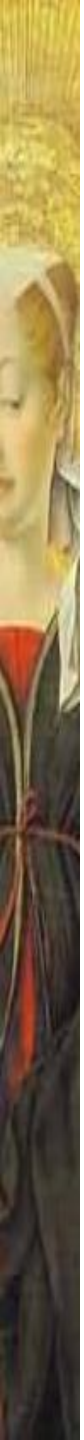


Debugging Tips

- Don’t panic!
- Find the first error message that refers to your code
- Move by small steps
- Implement the code from the slides first
- Proceed by slowly changing it

Lecture 2 Highlights

- Theano is functional
  - We run the Theano code once to create the model, and later train and test that model in a usual Python loop
- Theano heavily relies on numpy
- Most important function: *theano.tensor.grad()*

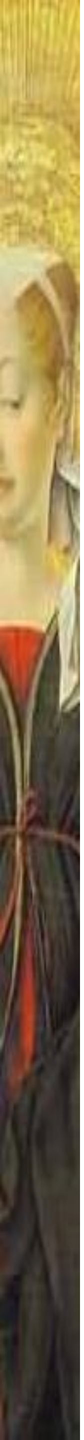


Big picture so far

- Learned: derivatives and gradients are crucial to train the models by optimizing their accuracy (reducing errors)
- Learned: Theano is a functional python library that efficiently obtains the gradients analytically
- Coming up: how we can use this to solve realistic scale problems

Plan for Today

- Training toy sentiment task in Theano
- Introduce neural networks
- Convert sentiment task to one
- Apply a neural network to realistic task: recognizing hand written digits



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!

Gradient Ascent in Theano

- Same dataset as in Lab 1
- Same linear function as in Lab 1
- But we want to split our ‘ratings’ array into two for convenience:
  - ‘texts’ will hold the review vectors
  - ‘score’ will hold the sentiment numbers (-2 to 2)

Split Data

```python
import theano
import numpy
texts = numpy.asarray([[1,1],[2,0],[1,0],[0,1],[0,1]])
scores = numpy.asarray([-2,2,0,1,-1])
```

- Note: we also use arrays from ‘numpy’
- This will allow us to use vector operations

Vector dot products

$$\begin{bmatrix} A_x & A_y & A_z \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = A_x B_x + A_y B_y + A_z B_z = \vec{A} \cdot \vec{B}$$

- To Note:
  - dot product of two vectors is a number (scalar)
  - It is a linear function with respect to each vector
  - So often used as convenient notation inside models

Weights to Train

```
import theano
import numpy
texts = numpy.asarray([[1,1],[2,0],[1,0],[0,1],[0,1]])
scores = numpy.asarray([-2,2,0,1,-1])

W = theano.shared(numpy.asarray([0.0, 0.0]), 'W') # The we
```

- The weights we used to call *w1* and *w2* are now two coordinates of the vector W
- This allows us to define our model as a dot product:

Linear Model

```
import theano
import numpy
texts = numpy.asarray([[1,1],[2,0],[1,0],[0,1],[0,1]])
scores = numpy.asarray([-2,2,0,1,-1])

W = theano.shared(numpy.asarray([0.0, 0.0]), 'W') # The weights for
predictions = theano.tensor.dot(texts, W) #this is our linear model
```

Accuracy

```
import theano
import numpy
texts = numpy.asarray([[1,1],[2,0],[1,0],[0,1],[0,1]])
scores = numpy.asarray([-2,2,0,1,-1])

W = theano.shared(numpy.asarray([0.0, 0.0]), 'W') # The weights for
predictions = theano.tensor.dot(texts, W) #this is our linear model
Accuracy = -theano.tensor.sqr(predictions - scores).sum() + 10
```

Things to Note:

- ‘sqr’ function is a vector operation
- For a vector of numbers $(x_1, x_2,\ldots,x_n)$ it creates a vector of squares of those numbers: $(x_1^2, x_2^2,\ldots,x_n^2)$
- ‘sum’ function adds up the coordinates
- This way we get the same definition of accuracy as in Lab 1
- We add 10 to be consistent with Lab 1

Gradients

```python
import theano
import numpy
texts = numpy.asarray([[1,1],[2,0],[1,0],[0,1],[0,1]])
scores = numpy.asarray([-2,2,0,1,-1])

W = theano.shared(numpy.asarray([0.0, 0.0]), 'W') # The weights for
predictions = theano.tensor.dot(texts, W) #this is our linear model
Accuracy = -theano.tensor.sqr(predictions - scores).sum() + 10
gradients = theano.tensor.grad(Accuracy, [W])
```

Updates

```
import theano
import numpy
texts = numpy.asarray([[1,1],[2,0],[1,0],[0
scores = numpy.asarray([-2,2,0,1,-1])

W = theano.shared(numpy.asarray([0.0, 0.0])
predictions = theano.tensor.dot(texts, W) #
Accuracy = -theano.tensor.sqr(predictions -
gradients = theano.tensor.grad(Accuracy, [W

W_updated = W + (0.1 * gradients[0])
updates = [(W, W_updated)]
```

Define Training Function

```
W_updated = W + (0.1 * gradients[0])
updates = [(W, W_updated)]

f = theano.function([], Accuracy, updates=updates)

for i in xrange(10):
    output = f()
    print output
```

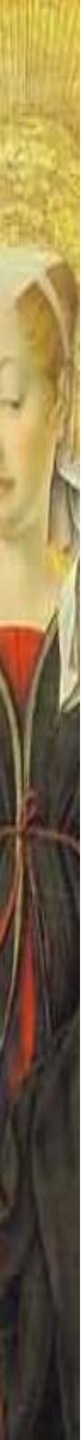
Now we can run!

```
W_updated = W + (0.1 * gradients[0])
updates = [(W, W_updated)]
f = theano.function([], Accuracy, updates=updates)
for i in xrange(10):
    output = f()
    print output
```

sentiment for slides | sentiment for slides

bugger | Console

```
Connected to pydev debugger (build 163.15188.4)
0.0
2.08
2.4832
2.566144
2.58356224
2.5872449536
2.58802529075
2.58819075408
2.58822584696
2.58823329029
```



Things to note:

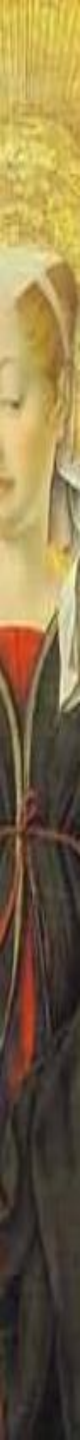
- Improving
- Stabilizing

High Level Picture

- We learned how gradient ascent can be implement to optimize accuracy of a model
  - (or reduce the errors by gradient descent)
- This means training a system
- So far we looked only at a model with a single linear function
- Real systems consist of thousands of such models with millions of parameters
- Now we are ready to start looking at more realistic models
- Specifically, Neural Networks

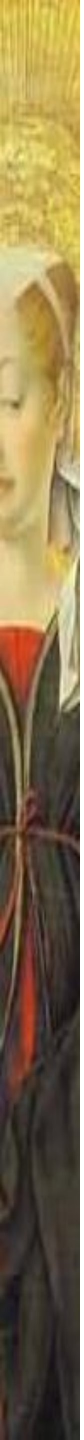
High Level Plan

- Keep using our toy Sentiment task
- Keep using Theano
- Convert our linear model to a neural network
  - By switching to stochastic gradient ascent
  - By predicting the score for each possible movie rating: we will now define W as 5x2 matrix
  - By applying non-linearity our predictions



Stochastic Gradient Ascent

- Computing gradient on entire data is impossible in realistic applications
- Thus, a certain number of data points is used to estimate it
- This approach is called Stochastic Gradient Ascent
- The number of points is called a mini-batch
- We will use minibatch size of 1
- So we will feed one training example at a time



Theano variables we will need

```
input_vector = theano.tensor.fvector('input_vector') #
target_values = theano.tensor.fvector('target_values')
```

- Note:
  - We will have to provide those two vectors to our Theano model on each training cycle
  - Theano will use *only* them to evaluate the accuracy and obtain its gradient

Dot product between matrix and a vector

$$\begin{pmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 1 & 4 & 2 \end{pmatrix}\begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1\cdot 3 + 1\cdot 1 + 1\cdot 2 \\ \\ \\ \end{pmatrix}$$

First row,

$$\begin{pmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 1 & 4 & 2 \end{pmatrix}\begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1\cdot 3 + 1\cdot 1 + 1\cdot 2 \\ 2\cdot 3 + 1\cdot 1 + 3\cdot 2 \\ \\ \end{pmatrix}$$

next row,

$$\begin{pmatrix} 1 & 1 & 2 \\ 2 & 1 & 3 \\ 1 & 4 & 2 \end{pmatrix}\begin{pmatrix} 3 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1\cdot 3 + 1\cdot 1 + 1\cdot 2 \\ 2\cdot 3 + 1\cdot 1 + 3\cdot 2 \\ 1\cdot 3 + 4\cdot 1 + 2\cdot 2 \end{pmatrix} = \begin{pmatrix} 6 \\ 13 \\ 11 \end{pmatrix}$$

last row,
then do the addition.

Re-Defining W as a matrix

```
input_vector = theano.tensor.fvector('input_vector') #the
target_values = theano.tensor.fvector('target_values') #t
W_initial_values = numpy.zeros((5,2))
W = theano.shared(W_initial_values, 'W')
```

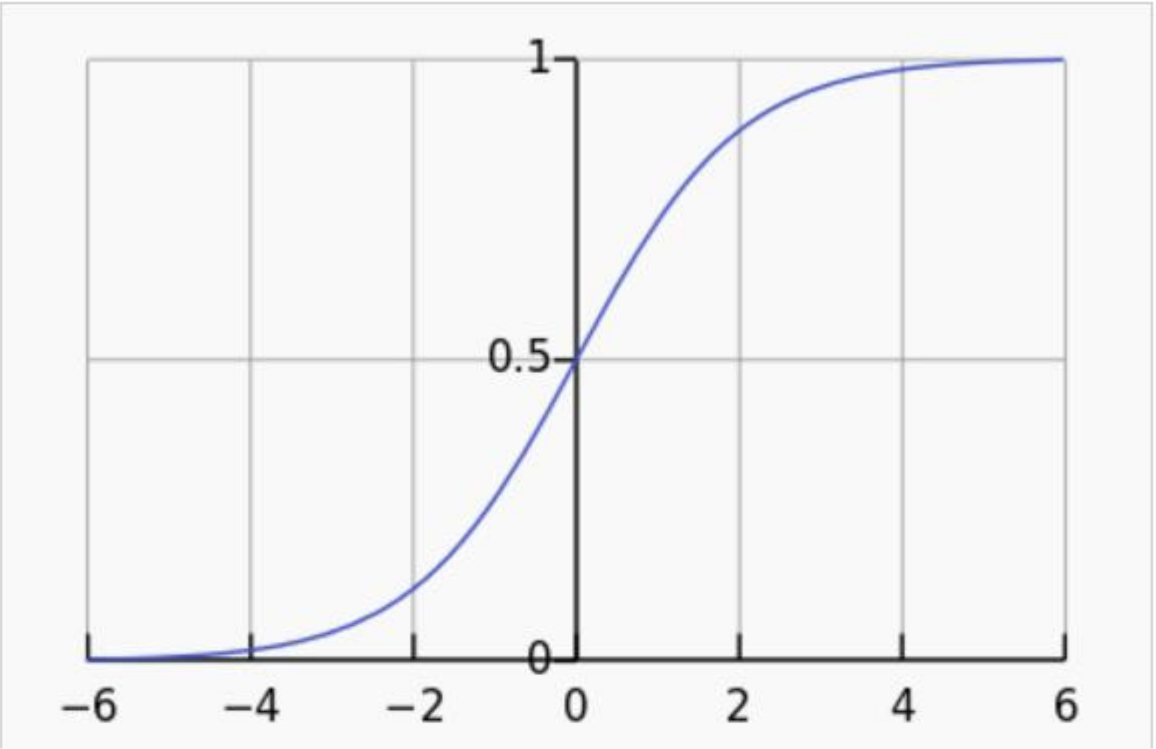
- To note:
  - W is a 5x2 matrix now
  - Its elements are all initialized to zero

Rename our linear predictions as “Activations”

```
input_vector = theano.tensor.fvector('input_vector') #the
target_values = theano.tensor.fvector('target_values') #t
W_initial_values = numpy.zeros((5,2))
W = theano.shared(W_initial_values, 'W')
activations = theano.tensor.dot(W, input_vector) #EXPLAI
```

- To Note:
  - “activations” is a vector of 5 numbers
  - One for each possible movie rating
  - we chose that name for future consistency with neural network terminology

Non linearity: sigmoid function



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

Applying non-linearity

```
W = theano.shared(numpy.zeros((5,2)), 'W')
activations = theano.tensor.dot(W, input_vector)
predicted_values = theano.tensor.nnet.sigmoid(activations)
```

- To notice:
  - Since ‘activations’ is a vector of 5 numbers
  - Predicted_values is also a vector of 5 numbers
  - To compute accuracy we need to compare them with actual values

One-hot encoding

- We have 5 possible review outcomes:

  [-2 very bad, -1 bad, 0 neutral, 1 good, 2 very good]
- This review score is our label
- The one we are trying to predict
- For convenience, we will encode each label as a vector of 5 numbers: all are 0s except the one corresponding to the correct label:

One-hot encoding

- very bad: [1,0,0,0,0]
- bad: [0,1,0,0,0]
- neutral: [0,0,1,0,0]
- good: [0,0,0,1,0]
- very good: [0,0,0,0,1]

Now we can define Accuracy

```
W = theano.shared(numpy.zeros((5,2)), 'W')
activations = theano.tensor.dot(W, input_vector)
predicted_values = theano.tensor.nnet.sigmoid(activations)
Accuracy = -theano.tensor.sqr(predicted_values - target_values).sum()
```

- We have seen that already, right?

And the predicted label

```
W = theano.shared(numpy.zeros((5,2)), 'W')
activations = theano.tensor.dot(W, input_vector)
predicted_values = theano.tensor.nnet.sigmoid(activations
Accuracy = -theano.tensor.sqr(predicted_values - target_v
predicted_class = theano.tensor.argmax(predicted_values)
```

- Note: you can also use a loop over all activations to find the max, but this is way simpler!

Add the usual gradient and update

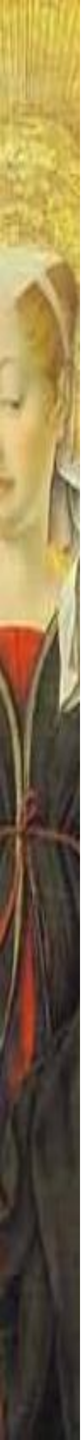
```
predicted_values = theano.tensor.nnet.sigmoid(activations)
Accuracy = -theano.tensor.sqr(predicted_values - target_values).su
predicted_class = theano.tensor.argmax(predicted_values)
gradients = theano.tensor.grad(Accuracy, W)
updates = [(W, W + .1 * gradients)]
```

Add the familiar function to train

```
gradients = theano.tensor.grad(Accuracy, W)
updates = [(W, W + .1 * gradients)]

train = theano.function([input_vector, target_values],
                        [W, activations, predicted_values, predicted_class, Accuracy, gradients],
                        updates=updates, allow_input_downcast=True)
```

- Note:
  - *[input_vector, target_values]* are inputs
  - The outputs is a long list starting with W
  - This is on purpose, so we can inspect them!
  - *updates=updates* – is standard Theano syntax to do updates
  - *allow_input_downcast=True* – also standard Theano syntax (will need it later!)



Let’s use some datapoint

- Review text: “This movie is just very good!“
- Rating: 2 (“very good”)
- Input vector?
- Target values?

Call ‘train’ and print the returns:

```
data_vector = [1., 0.]
target_vector = [0,0,0,0,1]
W, activations, predicted_values, predicted_class, Accuracy, gradients_W  = train(data_vector, target_vector)
print W, activations, predicted_values, predicted_class, Accuracy
print gradients_W
```

Run and check the output

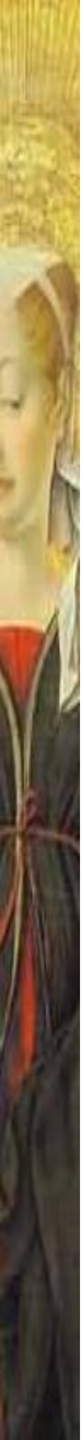
```
print W, activations, predicted_values, predicted_class, Accuracy
print gradients_W
```

L3 - toy for slides

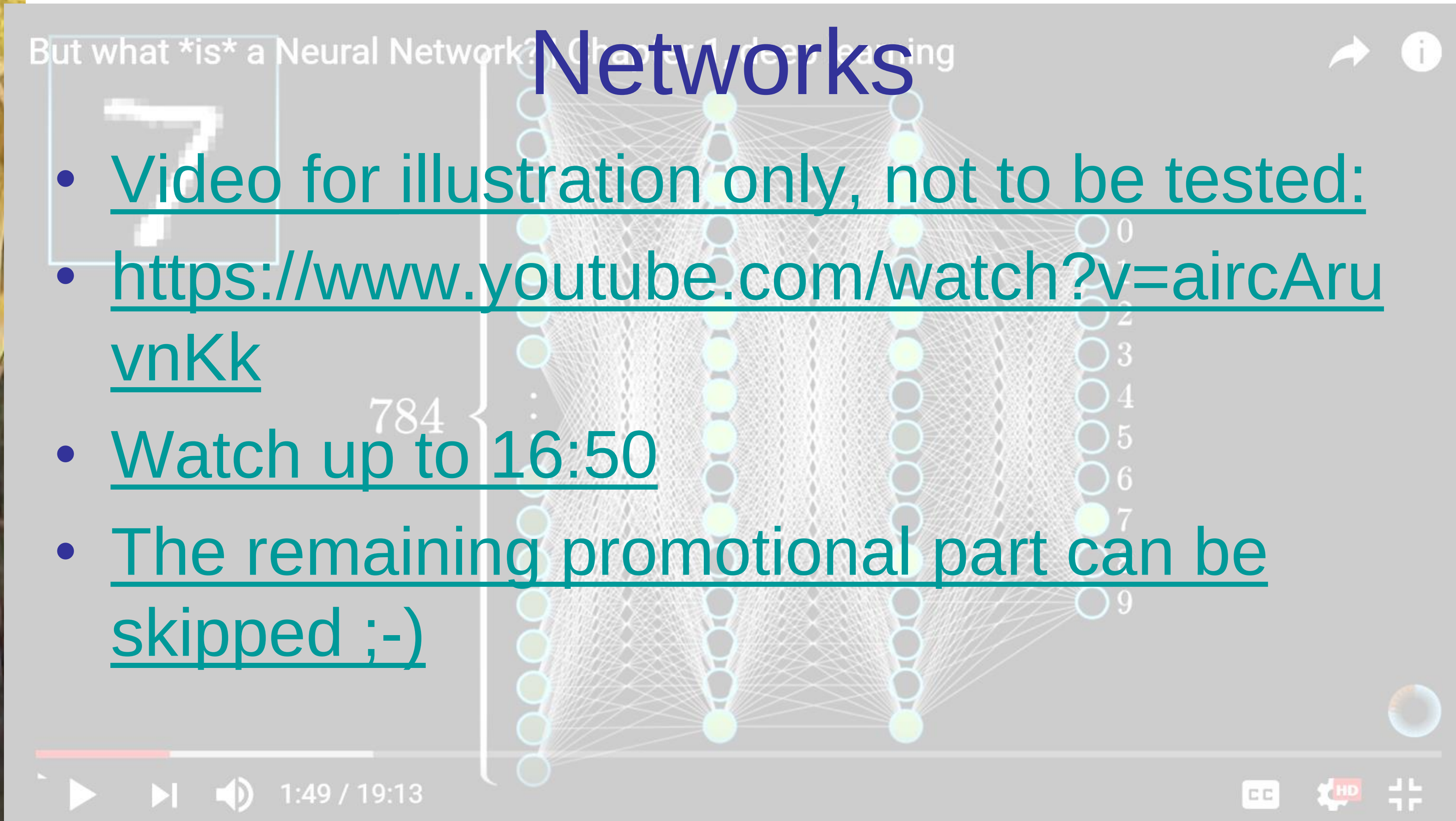
Console

```
pydev debugger: process 15864 is connecting

Connected to pydev debugger (build 163.15188.4)
[[ 0.   0.]
 [ 0.   0.]
 [ 0.   0.]
 [ 0.   0.]
 [ 0.   0.]] [ 0.   0.   0.   0.   0.] [ 0.5   0.5   0.5   0.5   0.5] 0 -1.25
[[-0.25 -0.  ]
 [-0.25 -0.  ]
 [-0.25 -0.  ]
 [-0.25 -0.  ]
 [ 0.25  0.  ]]
```



Introduction Into Neural Networks



- Video for illustration only, not to be tested:
- https://www.youtube.com/watch?v=aircAruvnKk
- Watch up to 16:50
- The remaining promotional part can be skipped ;-)