UNIVERSITY OF CAMBRIDGE

Deep Learning 101— a Hands-on Tutorial

Yarin Gal

yg279@cam.ac.uk

A TALK IN THREE ACTS, based in part on the online tutorial

deeplearning.net/software/theano/tutorial

Synopsis

”Deep Learning is not rocket science”

Why deep learning is so easy (in practice)

Playing with Theano

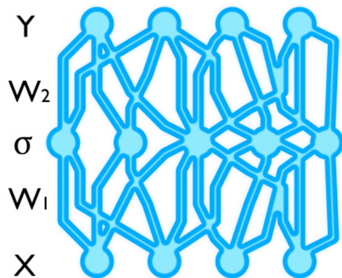
Two Theano examples: logistic regression and a deep net

Making deep learning even simpler: using existing packages

”Deep Learning is not rocket science”

Modern deep learning

UNIVERSITY OF CAMBRIDGE



Conceptually simple models...

- Attracts **tremendous attention** from popular media,
- **Fundamentally affected** the way ML is used in industry,
- Driven by **pragmatic** developments...
- of **tractable** models...
- that **work** well...
- and **scale** well.

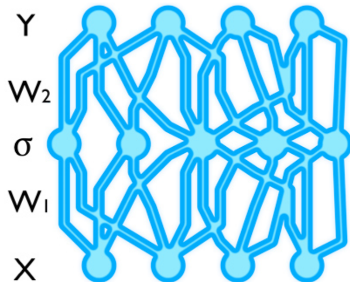
In more concrete terms...

UNIVERSITY OF CAMBRIDGE

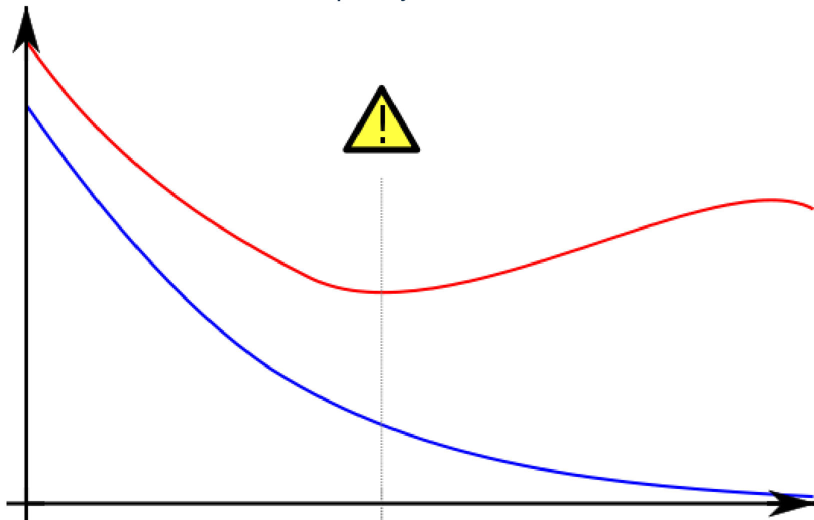
Data: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_N\}$
Model: given matrices **W** and non-linear func. $\sigma(\cdot)$, define "network"

$$\tilde{\mathbf{y}}_i(\mathbf{x}_i) = \mathbf{W}_2 \cdot \sigma\big(\mathbf{W}_1 \mathbf{x}_i\big)$$

Objective: find **W** for which $\tilde{\mathbf{y}}_i(\mathbf{x}_i)$ is close to $\mathbf{y}_i$ for all $i \leq N$.



- But these models overfit quickly...

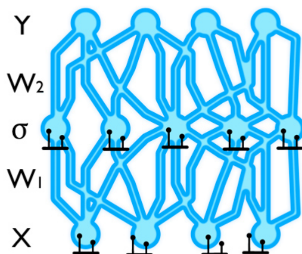


- Dropout is a technique to avoid overfitting:

Model regularisation in deep learning

UNIVERSITY OF CAMBRIDGE

- But these models overfit quickly...
- Dropout is a technique to avoid overfitting:
  - Used in **most modern deep learning models**



  - It circumvents **over-fitting** (we can discuss why later)
  - And improves **performance**

Example model: image processing

UNIVERSITY OF CAMBRIDGE

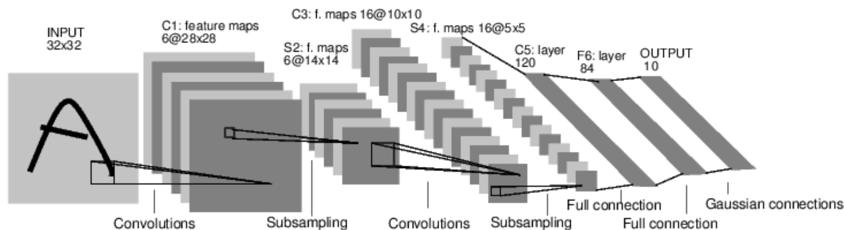


Figure: LeNet convnet structure

We'll see a concrete example later. But first, how do we find optimal weights **W easily**?

Why deep learning is so easy (in practice)

Symbolic differentiation

UNIVERSITY OF CAMBRIDGE

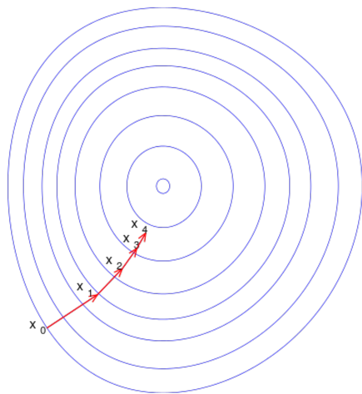
- Need to find optimal weights $\mathbf{W}_i$ minimising distance of model predictions $\tilde{\mathbf{y}}^{\mathbf{W}_1,\mathbf{W}_2}(\mathbf{x}_i) := \mathbf{W}_2 \cdot \sigma\big(\mathbf{W}_1\mathbf{x}_i\big)$ from observations $\mathbf{y}_i$

$$\mathcal{L}(\mathbf{W}_1, \mathbf{W}_2) = \sum_{i=1}^{N} (\mathbf{y}_i - \tilde{\mathbf{y}}^{\mathbf{W}_1,\mathbf{W}_2}(\mathbf{x}_i))^2 + \underbrace{||\mathbf{W}_1||^2 + ||\mathbf{W}_2||^2}_{\text{keeps weights from blowing up}}$$

$$\mathbf{W}_1, \mathbf{W}_2 = \text{argmin}_{\mathbf{W}_1,\mathbf{W}_2} \mathcal{L}(\mathbf{W}_1, \mathbf{W}_2)$$

- We can use calculus to differentiate objective $\mathcal{L}(\mathbf{W}_1, \mathbf{W}_2)$ w.r.t. $\mathbf{W}_1, \mathbf{W}_2$ and use *gradient descent*
- Differentiating $\mathcal{L}(\mathbf{W}_1, \mathbf{W}_2)$ is extremely easy using **symbolic differentiation**.

- Need to find optimal weights $\mathbf{W}_i$ minimising distance of model predictions $\tilde{\mathbf{y}}^{\mathbf{W}_1,\mathbf{W}_2}(\mathbf{x}_i) := \mathbf{W}_2 \cdot \sigma\big(\mathbf{W}_1 \mathbf{x}_i\big)$ from observations $\mathbf{y}_i$
- We can use calculus to differentiate objective $\mathcal{L}(\mathbf{W}_1, \mathbf{W}_2)$ w.r.t. $\mathbf{W}_1, \mathbf{W}_2$ and use *gradient descent*



Symbolic differentiation

UNIVERSITY OF CAMBRIDGE

- Need to find optimal weights $\mathbf{W}_i$ minimising distance of model predictions $\tilde{\mathbf{y}}^{\mathbf{W}_1,\mathbf{W}_2}(\mathbf{x}_i) := \mathbf{W}_2 \cdot \sigma\big(\mathbf{W}_1 \mathbf{x}_i\big)$ from observations $\mathbf{y}_i$
- We can use calculus to differentiate objective $\mathcal{L}(\mathbf{W}_1, \mathbf{W}_2)$ w.r.t. $\mathbf{W}_1, \mathbf{W}_2$ and use *gradient descent*
- Differentiating $\mathcal{L}(\mathbf{W}_1, \mathbf{W}_2)$ is extremely easy using **symbolic differentiation**.

What's symbolic differentiation?

UNIVERSITY OF CAMBRIDGE

- "Symbolic computation is a scientific area that refers to the study and development of algorithms and software for **manipulating mathematical expressions** and other mathematical objects." [Wikipedia]

What's Theano?

- Theano was the priestess of Athena in Troy [source: Wikipedia].
- It is *also* a **Python package for symbolic differentiation.**[a]
- Open source project primarily developed at the University of Montreal.
- Symbolic equations compiled to run efficiently on CPU and GPU.
- Computations are expressed using a NumPy-like syntax:
  - numpy.exp() – theano.tensor.exp()
  - numpy.sum() – theano.tensor.sum()

Figure: Athena

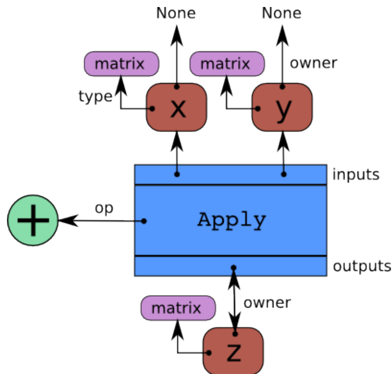
[a]TensorFlow (Google's Theano alternative) is similar.

UNIVERSITY OF CAMBRIDGE

Internally, Theano builds a graph structure composed of:

- interconnected variable nodes (red),
- operator (op) nodes (green),
- and "apply" nodes (blue, representing the application of an op to some variables)

```python
import theano.tensor as T
x = T.dmatrix('x')
y = T.dmatrix('y')
z = x + y
```



Computing automatic differentiation is simple with the graph structure.

- The only thing tensor.grad() has to do is to traverse the graph from the outputs back towards the inputs.
- Gradients are composed using the chain rule.

Code for derivatives of x^2:

```
1 x = T.scalar('x')
2 f = x**2
3 df_dx = T.grad(f, [x]) # results in 2x
```

Theano graph optimisation

UNIVERSITY OF CAMBRIDGE

When compiling a Theano graph, graph optimisation...

- Improves the way the computation is carried out,
- Replaces certain patterns in the graph with faster or more stable patterns that produce the same results,
- And detects identical sub-graphs and ensures that the same values are not computed twice (*mostly*).

For example, one optimisation is to replace the pattern $\frac{xy}{y}$ by x.

Playing with Theano

Theano in practice – example

UNIVERSITY OF CAMBRIDGE

```python
>>> import theano.tensor as T
>>> from theano import function
>>> x = T.dscalar('x')
>>> y = T.dscalar('y')
>>> z = x + y # same graph as before

>>> f = function([x, y], z) # compiling the graph
# the function inputs are x and y, its output is z
>>> f(2, 3) # evaluating the function on integers
array(5.0)
>>> f(16.3, 12.1) # ...and on floats
array(28.4)

>>> z.eval({x : 16.3, y : 12.1})
array(28.4) # a quick way to debug the graph

>>> from theano import pp
>>> print pp(z) # print the graph
(x + y)
```

UNIVERSITY OF CAMBRIDGE

1. Type and run the following code:

```
import theano
import theano.tensor as T
a = T.vector() # declare variable
out = a + a**10 # build symbolic expression
f = theano.function([a], out) # compile function
print f([0, 1, 2]) # prints 'array([0, 2, 1026])'
```

2. Modify the code to compute $a^2 + 2ab + b^2$ element-wise.

Theano basics – solution 1

UNIVERSITY OF CAMBRIDGE

```
1 import theano
2 import theano.tensor as T
3 a = T.vector()  # declare variable
4 b = T.vector()  # declare variable
5 out = a**2 + 2*a*b + b**2 # build symbolic expression
6 f = theano.function([a, b], out)    # compile function
7 print f([1, 2], [4, 5])  # prints [ 25.  49.]
```

Theano basics – exercise 2

Implement the *Logistic Function*:

$$s(x) = \frac{1}{1 + e^{-x}}$$

Theano basics – solution 2

UNIVERSITY OF CAMBRIDGE

```
1 >>> x = T.dmatrix('x')
2 >>> s = 1 / (1 + T.exp(-x))
3 >>> logistic = theano.function([x], s)
4 >>> logistic([[0, 1], [-1, -2]])
5 array([[ 0.5       ,  0.73105858],
6        [ 0.26894142,  0.11920292]])
```

Note that the operations are performed element-wise.

We can compute the elementwise *difference*, *absolute difference*, and *squared difference* between two matrices *a* and *b* at the same time.

```
1 >>> a, b = T.dmatrices('a', 'b')
2 >>> diff = a - b
3 >>> abs_diff = abs(diff)
4 >>> diff_squared = diff**2
5 >>> f = function([a, b], [diff, abs_diff, diff_squared])
```

Theano basics – shared variables

Shared variables allow for functions with internal states.

- hybrid symbolic and non-symbolic variables,
- value may be shared between multiple functions,
- used in symbolic expressions but also have an internal value.

The value can be accessed and modified by the .get_value() and .set_value() methods.

Accumulator

The state is initialized to zero. Then, on each function call, the state is incremented by the function's argument.

```
1 >>> state = theano.shared(0)
2 >>> inc = T.iscalar('inc')
3 >>> accumulator = theano.function([inc], state,
4                              updates=[(state, state+inc)])
```

Theano basics – updates parameter

- Updates can be supplied with a list of pairs of the form (shared-variable, new expression),
- Whenever function runs, it replaces the value of each shared variable with the corresponding expression's result at the end.

In the example above, the accumulator replaces *state*'s value with the sum of *state* and the increment amount.

```
>>> state.get_value()
array(0)
>>> accumulator(1)
array(0)
>>> state.get_value()
array(1)
>>> accumulator(300)
array(1)
>>> state.get_value()
array(301)
```

Two Theano examples: logistic regression and a deep net

- Logistic regression is a probabilistic linear classifier.
- It is parametrised by a weight matrix W and a bias vector b.
- The probability that an input vector x is classified as 1 can be written as:

$$P(Y = 1|x, W, b) = \frac{1}{1 + e^{-(Wx+b)}} = s(Wx + b)$$

- The model's prediction y_{pred} is the class whose probability is maximal, specifically for every x:

$$y_{pred} = 1(P(Y = 1|x, W, b) > 0.5)$$

- And the optimisation objective (negative log-likelihood) is

$$-y \log(s(Wx + b)) - (1 - y) \log(1 - s(Wx + b))$$

(you can put a Gaussian prior over W if you so desire.)

Using the Logistic Function, implement Logistic Regression.

```python
...
x = T.matrix("x")
y = T.vector("y")
w = theano.shared(np.random.randn(784), name="w")
b = theano.shared(0., name="b")

# Construct Theano expression graph
prediction, obj, gw, gb # Implement me!

# Compile
train = theano.function(inputs=[x,y],
          outputs=[prediction, obj],
          updates=((w, w - 0.1 * gw), (b, b - 0.1 * gb))
predict = theano.function(inputs=[x], outputs=prediction
          
# Train
for i in range(training_steps):
    pred, err = train(D[0], D[1])
...
```

Theano basics – solution 3

UNIVERSITY OF CAMBRIDGE

```
...
# Construct Theano expression graph
# Probability that target = 1
p_1 = 1 / (1 + T.exp(-T.dot(x, w) - b))
# The prediction thresholded
prediction = p_1 > 0.5
# Cross-entropy loss function
obj = -y * T.log(p_1) - (1-y) * T.log(1-p_1)
# The cost to minimize
cost = obj.mean() + 0.01 * (w ** 2).sum()
# Compute the gradient of the cost
gw, gb = T.grad(cost, [w, b])
...
```

Implement an MLP, following section *Example: MLP* in
`http://nbviewer.ipython.org/github/craffel/theano-tutorial/blob/master/Theano%20Tutorial.ipynb#example-mlp`

Theano basics – solution 4

UNIVERSITY OF CAMBRIDGE

```
1  class Layer(object):
2    def __init__(self, W_init, b_init, activation):
3      n_output, n_input = W_init.shape
4      self.W = theano.shared(value=W_init.astype(theano.co
5                             name='W',
6                             borrow=True)
7      self.b = theano.shared(value=b_init.reshape(-1, 1).a
8                             name='b',
9                             borrow=True,
10                            broadcastable=(False, True))
11     self.activation = activation
12     self.params = [self.W, self.b]
13
14   def output(self, x):
15     lin_output = T.dot(self.W, x) + self.b
16     return (lin_output if self.activation is None else s
```

```
1  class MLP(object):
2      def __init__(self, W_init, b_init, activations):
3          self.layers = []
4          for W, b, activation in zip(W_init, b_init, acti
5              self.layers.append(Layer(W, b, activation))
6
7          self.params = []
8          for layer in self.layers:
9              self.params += layer.params
10
11     def output(self, x):
12         for layer in self.layers:
13             x = layer.output(x)
14         return x
15
16     def squared_error(self, x, y):
17         return T.sum((self.output(x) - y)**2)
```

```
1  def gradient_updates_momentum(cost, params,
2      learning_rate, momentum):
3    updates = []
4    for param in params:
5      param_update = theano.shared(param.get_value()*0.,
6         broadcastable=param.broadcastable)
7      updates.append((param,
8        param - learning_rate*param_update))
9      updates.append((param_update, momentum*param_update
10       + (1. - momentum)*T.grad(cost, param)))
11   return updates
```

Making deep learning even simpler: using Keras

UNIVERSITY OF CAMBRIDGE

- **Keras** is a python package that uses Theano (or TensorFlow) to abstract away model design.
- A Sequential model is a **linear stack of layers**:

```
1  from keras.models import Sequential
2  from keras.layers import Dense, Activation
3
4  model = Sequential()
5  model.add(Dense(32, input_dim=784))
6  model.add(Activation('relu'))
7
8  # for a mean squared error regression problem
9  model.compile(optimizer='rmsprop', loss='mse')
10
11 # train the model
12 model.fit(X, Y, nb_epoch=10, batch_size=32)
```

- **Keras** is a python package that uses Theano (or TensorFlow) to abstract away model design.
- A Sequential model is a **linear stack of layers**:

```
1  from keras.models import Sequential
2  from keras.layers import Dense, Activation
3
4  model = Sequential()
5  model.add(Dense(32, input_dim=784))
6  model.add(Activation('relu'))
7
8  # for a mean squared error regression problem
9  model.compile(optimizer='rmsprop', loss='mse')
10
11 # train the model
12 model.fit(X, Y, nb_epoch=10, batch_size=32)
```

UNIVERSITY OF CAMBRIDGE

Follow tutorial on `http://goo.gl/xatlXR`

In your free time:

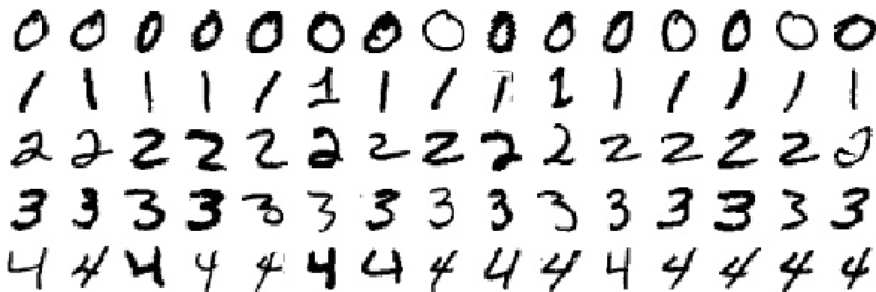


Image processing example on `https://goo.gl/G4ccHU`

Thank you

UNIVERSITY OF CAMBRIDGE