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
"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

- 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.

*Conceptually simple models*...
In more concrete terms...

**Data:** $X = \{x_1, x_2, \ldots, x_N\}$, $Y = \{y_1, y_2, \ldots, y_N\}$

**Model:** given matrices $W$ and non-linear func. $\sigma(\cdot)$, define “network”

$$\tilde{y}_i(x_i) = W_2 \cdot \sigma(W_1 x_i)$$

**Objective:** find $W$ for which $\tilde{y}_i(x_i)$ is close to $y_i$ for all $i \leq N$. 
But these models overfit quickly...

Dropout is a technique to avoid overfitting:
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
We’ll see a concrete example later. But first, how do we find optimal weights $\mathbf{W}$ easily?
Why deep learning is so easy (in practice)
Need to find optimal weights $W_i$ minimising distance of model predictions $\tilde{y}^{W_1,W_2}(x_i) := W_2 \cdot \sigma(W_1 x_i)$ from observations $y_i$

$$
\mathcal{L}(W_1, W_2) = \sum_{i=1}^{N} (y_i - \tilde{y}^{W_1,W_2}(x_i))^2 + \|W_1\|^2 + \|W_2\|^2
$$
keeps weights from blowing up

$$
W_1, W_2 = \text{argmin}_{W_1,W_2} \mathcal{L}(W_1, W_2)
$$

We can use calculus to differentiate objective $\mathcal{L}(W_1, W_2)$ w.r.t. $W_1, W_2$ and use gradient descent.

Differentiating $\mathcal{L}(W_1, W_2)$ is extremely easy using symbolic differentiation.
Symbolic differentiation

- Need to find optimal weights $W_i$ minimising distance of model predictions $\tilde{y}^{W_1, W_2}(x_i) := W_2 \cdot \sigma(W_1 x_i)$ from observations $y_i$

- We can use calculus to differentiate objective $L(W_1, W_2)$ w.r.t. $W_1, W_2$ and use gradient descent
Need to find optimal weights $W_i$ minimising distance of model predictions $\tilde{y}^{W_1,W_2}(x_i) := W_2 \cdot \sigma(W_1 x_i)$ from observations $y_i$.

We can use calculus to differentiate objective $\mathcal{L}(W_1, W_2)$ w.r.t. $W_1, W_2$ and use gradient descent.

Differentiating $\mathcal{L}(W_1, W_2)$ is extremely easy using **symbolic differentiation**.
What’s symbolic differentiation?

▸ “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()`

\(^a\)TensorFlow (Google’s Theano alternative) is similar.
How does Theano work?

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
1  import theano.tensor as T
2  x = T.dmatrix('x')
3  y = T.dmatrix('y')
4  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 \( x^2 \):**

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

```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)
```
1. Type and run the following code:

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

2. Modify the code to compute $a^2 + 2ab + b^2$ element-wise.
import theano
import theano.tensor as T
a = T.vector() # declare variable
b = T.vector() # declare variable
out = a**2 + 2*a*b + b**2 # build symbolic expression
f = theano.function([a, b], out) # compile function
print f([1, 2], [4, 5]) # prints [ 25.  49.]
Implement the *Logistic Function*:

\[
s(x) = \frac{1}{1 + e^{-x}}
\]
Theano basics – solution 2

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.

```python
>>> a, b = T.dmatrices('a', 'b')
>>> diff = a - b
>>> abs_diff = abs(diff)
>>> diff_squared = diff**2
>>> 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.

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

```python
>>> state.get_value()
aarray(0)
>>> accumulator(1)
aarray(0)
>>> state.get_value()
aarray(1)
>>> accumulator(300)
aarray(1)
>>> state.get_value()
aarray(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.
...  
1. \( x = T.\text{matrix}("x") \)  
2. \( y = T.\text{vector}("y") \)  
3. \( w = \text{theano.\text{shared}}(\text{np.random.randn}(784), \text{name}="w") \)  
4. \( b = \text{theano.\text{shared}}(0., \text{name}="b") \)  

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

# Compile
6. train = theano.\text{function}(\text{inputs}=[x, y],  
   \quad \text{outputs}=[\text{prediction}, \text{obj}],  
   \quad \text{updates}=((w, w - 0.1 \ast gw), (b, b - 0.1 \ast gb)))  
7. predict = theano.\text{function}(\text{inputs}=[x], \text{outputs}=\text{prediction})  

# Train
8. \text{for} i \text{ in} \text{range}(\text{training\_steps}):  
   \quad \text{pred, err} = \text{train}(D[0], D[1])  
9. ...
... 

# 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
class Layer(object):
    def __init__(self, W_init, b_init, activation):
        n_output, n_input = W_init.shape
        self.W = theano.shared(value=W_init.astype(theano.config.floatX),
                               name='W',
                               borrow=True)
        self.b = theano.shared(value=b_init.reshape(-1, 1).astype(theano.config.floatX),
                               name='b',
                               borrow=True,
                               broadcastable=(False, True))

        self.activation = activation
        self.params = [self.W, self.b]

    def output(self, x):
        lin_output = T.dot(self.W, x) + self.b
        return (lin_output if self.activation is None else self.activation(lin_output))
class MLP(object):
    def __init__(self, W_init, b_init, activations):
        self.layers = []
        for W, b, activation in zip(W_init, b_init, activations):
            self.layers.append(Layer(W, b, activation))

        self.params = []
        for layer in self.layers:
            self.params += layer.params

    def output(self, x):
        for layer in self.layers:
            x = layer.output(x)
        return x

    def squared_error(self, x, y):
        return T.sum((self.output(x) - y)**2)
```python
def gradient_updates_momentum(cost, params,
                            learning_rate, momentum):
    updates = []
    for param in params:
        param_update = theano.shared(param.get_value() * 0.,
                                      broadcastable=param.broadcastable)
        updates.append(((param,
                           param - learning_rate*param_update))
        updates.append(((param_update, momentum*param_update
                           + (1. - momentum)*T.grad(cost, param)))
    return updates
```
Making deep learning even simpler: using Keras
Keras is a python package that uses Theano (or TensorFlow) to abstract away model design.

A Sequential model is a linear stack of layers:

```python
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))

# for a mean squared error regression problem
model.compile(optimizer='rmsprop', loss='mse')

# train the model
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**:

```python
from keras.models import Sequential
from keras.layers import Dense, Activation

model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))

# for a mean squared error regression problem
model.compile(optimizer='rmsprop', loss='mse')

# train the model
model.fit(X, Y, nb_epoch=10, batch_size=32)
```
Follow tutorial on [http://goo.gl/xatlXR](http://goo.gl/xatlXR)

In your free time:

Image processing example on [https://goo.gl/G4ccHU](https://goo.gl/G4ccHU)