UNIVERSITY OF CAMBRIDGE

Symbolic Differentiation for Rapid Model Prototyping in Machine Learning and Data Analysis — a Hands-on Tutorial

Yarin Gal

yg279@cam.ac.uk

November 13th, 2014

A TALK IN TWO ACTS, based on the online tutorial

deeplearning.net/software/theano/tutorial

Outline

The Theory

Theano in practice

Two Example Models: Logistic Regression and a Deep Net

Rapid Prototyping of Probabilistic Models with SVI (time permitting)

UNIVERSITY OF CAMBRIDGE

Some Theory

What's symbolic differentiation?

UNIVERSITY OF CAMBRIDGE

- Symbolic differentiation is *not* automatic differentiation, nor numerical differentiation [source: Wikipedia].
- 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.

What's Theano?

- Theano was the priestess of Athena in Troy [source: Wikipedia].
- It is *also* a **Python package for symbolic differentiation**.
- 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

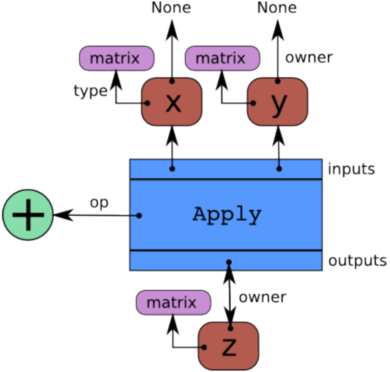
How does Theano work?

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
```



Theano basics – differentiation

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:

```
x = T.scalar('x')
f = x**2
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.

The Practice

Theano in practice – example

```
>>> 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)
```

Theano in practice – note

If you don't have Theano installed, you can SSH into one of the following computers and use the Python console:

- riemann
- dirichlet
- bernoulli
- grothendieck
- robbins
- explorer

Syntax (from an external network):

```
ssh [user name]@gate.eng.cam.ac.uk
ssh [computer name]
python
>>> import theano
>>> import theano.tensor as T
```

Exercise files are on `http://goo.gl/r5uwGI`

Theano basics – exercise 1

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

```
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*:

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

(adapt your NumPy implementation, you will need to replace “np” with “T”; this will be used later in Logistic regression)

Theano basics – solution 2

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

Note that the operations are performed element-wise.

Theano basics – multiple inputs outputs

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

```
>>> 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.

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

UNIVERSITY OF CAMBRIDGE

- 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)
```

UNIVERSITY OF CAMBRIDGE

Two Example Models: 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:

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

- 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.

UNIVERSITY OF CAMBRIDGE

```
...
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

```
...
# 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

```
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.co
                           name='W',
                           borrow=True)
    self.b = theano.shared(value=b_init.reshape(-1, 1).a
                           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 s
```

Theano basics – solution 4


```python
class MLP(object):
    def __init__(self, W_init, b_init, activations):
        self.layers = []
        for W, b, activation in zip(W_init, b_init, acti
            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)
```

Theano basics – solution 4

```
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
```

Rapid Prototyping of Probabilistic Models with Stochastic Variational Inference

- In **data analysis** we often have to develop new models
- This can be a lengthy process
  - We need to derive appropriate inference
  - Often cumbersome implementation which changes regularly
- **Rapid prototyping** is used to answer similar problems in manufacturing
  - “Quick fabrication of scale models of a physical part”
  - Probabilistic programming can be used for rapid prototyping in machine learning

- In **data analysis** we often have to develop new models
- This can be a lengthy process
  - We need to derive appropriate inference
  - Often cumbersome implementation which changes regularly
- **Rapid prototyping** is used to answer similar problems in manufacturing
  - "Quick fabrication of scale models of a physical part"
  - Probabilistic programming can be used for rapid prototyping in machine learning

Rapid Prototyping

- In **data analysis** we often have to develop new models
- This can be a lengthy process
  - We need to derive appropriate inference
  - Often cumbersome implementation which changes regularly
- **Rapid prototyping** is used to answer similar problems in manufacturing
  - “Quick fabrication of scale models of a physical part”
  - Probabilistic programming can be used for rapid prototyping in machine learning

Rapid Prototyping

- In **data analysis** we often have to develop new models
- This can be a lengthy process
  - We need to derive appropriate inference
  - Often cumbersome implementation which changes regularly
- **Rapid prototyping** is used to answer similar problems in manufacturing
  - "Quick fabrication of scale models of a physical part"
  - Probabilistic programming can be used for rapid prototyping in machine learning

Stochastic Variational Inference (SVI) can be used for rapid prototyping as well, with several advantages over probabilistic programming.

- SVI is not usually considered as means of speeding-up development
- But this new inference technique allows us to **simplify the derivations** for a large class of models
  - With this we can take advantage of **effective symbolic differentiation**
  - Models are often mathematically **too cumbersome otherwise**
- Similar principles have been used for **rapid model prototyping in deep learning** for NLP for quite some time [Socher, Ng, and Manning 2010, 2011, 2012]

What is SVI?

UNIVERSITY OF CAMBRIDGE

- SVI is simply **variational inference** used with **noisy gradients** – we thus replace the optimisation with stochastic optimisation
- Variational inference
  - We approximate the posterior of the latent variables with distributions from a tractable family ($q(X)$ for example)

Example model: $X \rightarrow Y$

$$\log P(Y) \geq \int q(X) \log \frac{P(Y|X)P(X)}{q(X)} = E_q[\log P(Y|X)] - KL(q||P)$$

What is SVI?

- Stochastic variational inference
  - Often used to speed-up inference using **mini-batches**

$$\log P(Y) \geq \frac{N}{|S|} \sum_{i \in S} E_q[\log P(Y_i | X_i)] - KL(q||P)$$

summing over random subsets of the data points

  - But can also be used to **approximate integrals** through Monte Carlo integration [Kingma and Welling 2014, Rezende et al. 2014, Titsias and Lazaro-Gredilla 2014]

$$E_q[\log P(Y|X)] \approx \frac{1}{K} \sum_{i=1}^{K} \log P(Y|X_i), \ X_i \sim q(X)$$

summing over samples from the approximating distribution

  - Optimising these objectives relies on non-deterministic gradients

What is SVI?

- Stochastic variational inference
  - Often used to speed-up inference using **mini-batches**

$$\log P(Y) \geq \frac{N}{|S|} \sum_{i \in S} E_q[\log P(Y_i | X_i)] - KL(q || P)$$

summing over random subsets of the data points

  - But can also be used to **approximate integrals** through Monte Carlo integration [Kingma and Welling 2014, Rezende et al. 2014, Titsias and Lazaro-Gredilla 2014]

$$E_q[\log P(Y|X)] \approx \frac{1}{K} \sum_{i=1}^{K} \log P(Y|X_i), \ X_i \sim q(X)$$

summing over samples from the approximating distribution

  - Optimising these objectives relies on non-deterministic gradients

What is SVI?

- Stochastic variational inference
  - Often used to speed-up inference using **mini-batches**

$$\log P(Y) \geq \frac{N}{|S|} \sum_{i \in S} E_q[\log P(Y_i | X_i)] - KL(q||P)$$

    summing over random subsets of the data points

  - But can also be used to **approximate integrals** through Monte Carlo integration [Kingma and Welling 2014, Rezende et al. 2014, Titsias and Lazaro-Gredilla 2014]

$$E_q[\log P(Y|X)] \approx \frac{1}{K} \sum_{i=1}^{K} \log P(Y|X_i), \; X_i \sim q(X)$$

    summing over samples from the approximating distribution

  - Optimising these objectives relies on non-deterministic gradients

Stochastic optimisation

- Using gradient descent with noisy gradients and decreasing learning-rates, we are guaranteed to converge to an optimum

$$\theta_{t+1} = \theta_t + \alpha f'(\theta_t)$$

- Learning-rates (α) are hard to tune...
  - Use learning-rate free optimisation (again, from deep learning)
  - AdaGrad [Duchi et. al 2011], AdaDelta [Zeiler 2012]
  - RMSPROP [Tieleman and Hinton 2012, Lecture 6.5, COURSERA: Neural Networks for Machine Learning]

$$\theta_{t+1} = \theta_t + \frac{\alpha}{\sqrt{r_t}} f'(\theta_t); \; r_t = (1-\gamma) f'(\theta)^2 + \gamma r_{t-1}$$

    and increase α times $1+\epsilon$ if the last two grads' directions agree
  - These have been compared to each other and others *empirically* in a variety of settings in [Schaul 2014]

Stochastic optimisation

- Using gradient descent with noisy gradients and decreasing learning-rates, we are guaranteed to converge to an optimum

$$\theta_{t+1} = \theta_t + \alpha f'(\theta_t)$$

- Learning-rates (α) are hard to tune...
  - Use learning-rate free optimisation (again, from deep learning)
  - AdaGrad [Duchi et. al 2011], AdaDelta [Zeiler 2012]
  - RMSPROP [Tieleman and Hinton 2012, Lecture 6.5, COURSERA: Neural Networks for Machine Learning]

$$\theta_{t+1} = \theta_t + \frac{\alpha}{\sqrt{r_t}} f'(\theta_t); \;\; r_t = (1-\gamma) f'(\theta)^2 + \gamma r_{t-1}$$

    and increase α times $1+\epsilon$ if the last two grads' directions agree

  - These have been compared to each other and others *empirically* in a variety of settings in [Schaul 2014]

Stochastic optimisation

- Using gradient descent with noisy gradients and decreasing learning-rates, we are guaranteed to converge to an optimum

$$\theta_{t+1} = \theta_t + \alpha f'(\theta_t)$$

- Learning-rates (α) are hard to tune...
  - Use learning-rate free optimisation (again, from deep learning)
  - AdaGrad [Duchi et. al 2011], AdaDelta [Zeiler 2012]
  - RMSPROP [Tieleman and Hinton 2012, Lecture 6.5, COURSERA: Neural Networks for Machine Learning]

$$\theta_{t+1} = \theta_t + \frac{\alpha}{\sqrt{r_t}} f'(\theta_t);\ r_t = (1-\gamma) f'(\theta)^2 + \gamma r_{t-1}$$

    and increase α times $1+\epsilon$ if the last two grads' directions agree
  - These have been compared to each other and others *empirically* in a variety of settings in [Schaul 2014]

Stochastic optimisation

- Using gradient descent with noisy gradients and decreasing learning-rates, we are guaranteed to converge to an optimum

$$\theta_{t+1} = \theta_t + \alpha f'(\theta_t)$$

- Learning-rates (α) are hard to tune...
  - Use learning-rate free optimisation (again, from deep learning)
  - AdaGrad [Duchi et. al 2011], AdaDelta [Zeiler 2012]
  - RMSPROP [Tieleman and Hinton 2012, Lecture 6.5, COURSERA: Neural Networks for Machine Learning]

$$\theta_{t+1} = \theta_t + \frac{\alpha}{\sqrt{r_t}} f'(\theta_t);\ r_t = (1-\gamma) f'(\theta)^2 + \gamma r_{t-1}$$

  and increase α times $1+\epsilon$ if the last two grads' directions agree

  - These have been compared to each other and others *empirically* in a variety of settings in [Schaul 2014]

Stochastic optimisation

- Using gradient descent with noisy gradients and decreasing learning-rates, we are guaranteed to converge to an optimum

$$\theta_{t+1} = \theta_t + \alpha f'(\theta_t)$$

- Learning-rates (α) are hard to tune...
  - Use learning-rate free optimisation (again, from deep learning)
  - AdaGrad [Duchi et. al 2011], AdaDelta [Zeiler 2012]
  - RMSPROP [Tieleman and Hinton 2012, Lecture 6.5, COURSERA: Neural Networks for Machine Learning]

$$\theta_{t+1} = \theta_t + \frac{\alpha}{\sqrt{r_t}} f'(\theta_t); \; r_t = (1-\gamma) f'(\theta)^2 + \gamma r_{t-1}$$

    and increase α times $1+\epsilon$ if the last two grads' directions agree
  - These have been compared to each other and others *empirically* in a variety of settings in [Schaul 2014]

Rapid Prototyping with SVI

With **Monte Carlo integration** we can greatly simplify model and inference description

Example model: X $\rightarrow$ Y

Lower bound:

1. Simulate $X_i \sim q(X)$ for $i \leq K$
2. Evaluate $P(Y|X_i)$
3. Return $\frac{1}{K}\sum_{i=1}^{K} \log P(Y|X_i) - KL(q||P)$

Objective:

$$q_{opt} = \arg\max_{q(X)} \frac{1}{K}\sum_{i=1}^{K} \log P(Y|X_i) - KL(q||P)$$

Example model: X $\rightarrow$ Y

Objective:

$$q_{opt} = \arg\max_{q(X)} \frac{1}{K} \sum_{i=1}^{K} \log P(Y|X_i) - KL(q||P)$$

Symbolic differentiation is straight-forward in this representation:

$$\frac{\partial}{\partial\theta} \log P(Y|X), \ \frac{\partial}{\partial\theta} KL$$

are easy to compute for a large class of models [Titsias and Lazaro-Gredilla 2014]

Examples: Bayesian logistic regression, variable selection, Gaussian process (GP) hyper-parameter estimation, and more [Titsias and Lazaro-Gredilla 2014]

Example: Bayesian logistic regression

Given dataset with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$ for $n \leq N$, we define

$$P(Y|X, \eta) = \prod_{i=1}^{N} \sigma(y_i \mathbf{x}_i^T \eta)$$

for some vector of weights η with prior $P(\eta) = \mathcal{N}(0, I_d)$.
Define

$$q(\eta|\theta = \{\mu, C\}) = \mathcal{N}(\eta; \mu, CC^T)$$

Symbolically differentiate and optimise wrt

$$\frac{\partial}{\partial \theta} \log \big(\prod_{i=1}^{N} \sigma(y_i \mathbf{x}_i^T \eta) \big), \; \frac{\partial}{\partial \theta} KL$$

UNIVERSITY OF CAMBRIDGE

Non-linear density estimation of categorical data (work in progress with Yutian Chen)

Model (using sparse GP with M inducing inputs / outputs Z and U):

$$X \sim \mathcal{N}(0, I)$$
$$(F_K, U_K) \sim GP(X, Z)$$
$$Y \sim Softmax(F_1, ..., F_K)$$

Approximating distributions: $q(X, F, U) = q(X)q(U)p(F|X, U)$, defining $q(x_n) = \mathcal{N}(m_n, s_n^2)$ and $q(u_k) = \mathcal{N}(\mu_k, CC^T)$

We have (with $\epsilon_{\cdot} \sim \mathcal{N}(0, I)$):

$$x_n = m_n + s_n \epsilon_n$$
$$u_k = \mu_k + C\epsilon_k$$
$$f_{nk} = K_{nM} K_{MM}^{-1} u_k + \sqrt{K_{nn} - K_{nM} K_{MM}^{-1} K_{Mn}} \epsilon_{nk}$$
$$y_n = Softmax(f_{n1}, ..., f_{nK})$$

Concrete example

- Original approach took half a year to develop –
  - Deriving variational inference
  - Researching appropriate bound in the statistics literature
  - Derivations for the model

$$
\begin{aligned}
\mathcal{L} = &-\sum_{n=1}^{N} \mathrm{KL}(q(\mathbf{x}_n) \| p(\mathbf{x}_n)) \\
&+ \sum_{d=1}^{D} \Big\{ -\frac{1}{2}\mathbf{Tr}[(I_K \otimes \mathbf{K}_{d,MM}^{-1})(\boldsymbol{\Sigma}_d + \hat{\boldsymbol{\mu}}_d \hat{\boldsymbol{\mu}}_d^T)] + \frac{MK}{2} + \frac{1}{2}\log|\boldsymbol{\Sigma}_d| - \frac{1}{2}\log|\mathbf{K}_{d,MM}| \Big\} \\
&+ \sum_{d=1}^{D}\sum_{n=1}^{N} \Big\{ -\frac{1}{2}\mathbf{Tr}\Big[\Big(\mathbf{A}_d \otimes (\mathbf{K}_{d,MM}^{-1} \langle \mathbf{K}_{d,Mn}\mathbf{K}_{d,nM}\rangle_{\mathbf{x}_n} \mathbf{K}_{d,MM}^{-1})\Big)(\boldsymbol{\Sigma}_d + \hat{\boldsymbol{\mu}}_d \hat{\boldsymbol{\mu}}_d^T)\Big] \\
&+ [(\mathbf{y}_{nd} + \mathbf{b}_{nd})^T \otimes (\langle \mathbf{K}_{d,nM}\rangle_{\mathbf{x}_n} \mathbf{K}_{d,MM}^{-1})]\hat{\boldsymbol{\mu}}_d - c_{nd} - \frac{1}{2}\mathbf{Tr}[\mathbf{A}_d \langle \mathbf{K}_{d,nn}\rangle_{\mathbf{x}_n}] + \frac{1}{2}\mathbf{Tr}(\mathbf{A}_d)\mathbf{Tr}(\mathbf{K} \\
= &-\sum_{n=1}^{N} \mathrm{KL}(q(\mathbf{x}_n) \| p(\mathbf{x}_n)) \\
&+ \sum_{d=1}^{D} \Big\{ -\frac{1}{2}\mathbf{Tr}[(I_K \otimes \mathbf{K}_{MM}^{-1})(\boldsymbol{\Sigma} + \hat{\boldsymbol{\mu}} \hat{\boldsymbol{\mu}}^T)] + \frac{MK}{2} + \frac{1}{2}\log|\boldsymbol{\Sigma}| - \frac{1}{2}\log|\mathbf{K}_{MM}|
\end{aligned}
$$

  - Implementation (hundreds of lines of python code)

- New approach –
  - Derivations took a day
  - Programming took a day (15 lines of Python)

Concrete example

UNIVERSITY OF CAMBRIDGE

- Original approach took half a year to develop –
  - Deriving variational inference
  - Researching appropriate bound in the statistics literature
  - Derivations for the model
  - Implementation (hundreds of lines of python code)

- New approach –
  - Derivations took a day
  - Programming took a day (15 lines of Python)

```
print 'Building model...'
X = m + T.exp(s) * eps_NQ
U = mu + T.tril(L).dot(eps_MK)

dist_ZZ = T.sum(((T.reshape(Z, (M, 1, Q)) - Z) / ard)**2, 2)
dist_ZX = T.sum(((T.reshape(Z, (M, 1, Q)) - X) / ard)**2, 2)
Kmm = sf2 * T.exp(-dist_ZZ / 2.0)
Kmn = sf2 * T.exp(-dist_ZX / 2.0)
Knn = sf2

KmmInv = sT.matrix_inverse(Kmm)
A = KmmInv.dot(Kmn)
B = Knn - T.sum(Kmn * KmmInv.dot(Kmn), 0)

F = A.T.dot(U) + B[:, None]**0.5 * eps_NK
S = T.nnet.softmax(F)
```

Disadvantages of this approach

UNIVERSITY OF CAMBRIDGE

- Studying how symbolic differentiation works is important though –
  - Careless implementation can take long to run
  - But careful implementation (together with mini batches) can actually scale well!
- Only suitable when variational inference is; As usual in variational inference depends on the family of approximating distributions
- We can have large variance in the approximate integration
  - Either use more samples (slower to run),
  - Or use variance reduction techniques [Wang, Chen, Smola, and Xing 2013]

Disadvantages of this approach

- Studying how symbolic differentiation works is important though –
  - Careless implementation can take long to run
  - But careful implementation (together with mini batches) can actually scale well!
- Only suitable when variational inference is; As usual in variational inference depends on the family of approximating distributions
- We can have large variance in the approximate integration
  - Either use more samples (slower to run),
  - Or use variance reduction techniques [Wang, Chen, Smola, and Xing 2013]

Disadvantages of this approach

UNIVERSITY OF CAMBRIDGE

- Studying how symbolic differentiation works is important though –
  - Careless implementation can take long to run
  - But careful implementation (together with mini batches) can actually scale well!
- Only suitable when variational inference is; As usual in variational inference depends on the family of approximating distributions
- We can have large variance in the approximate integration
  - Either use more samples (slower to run),
  - Or use variance reduction techniques [Wang, Chen, Smola, and Xing 2013]

Thank you

UNIVERSITY OF CAMBRIDGE