Tutorial: deep learning & Theano

Sander Dieleman

February 10th, 2015

PhD student at the Reservoir Lab, Ghent University

Working on audio-based music classification, recommendation, …

Deep learning, feature learning, neural networks

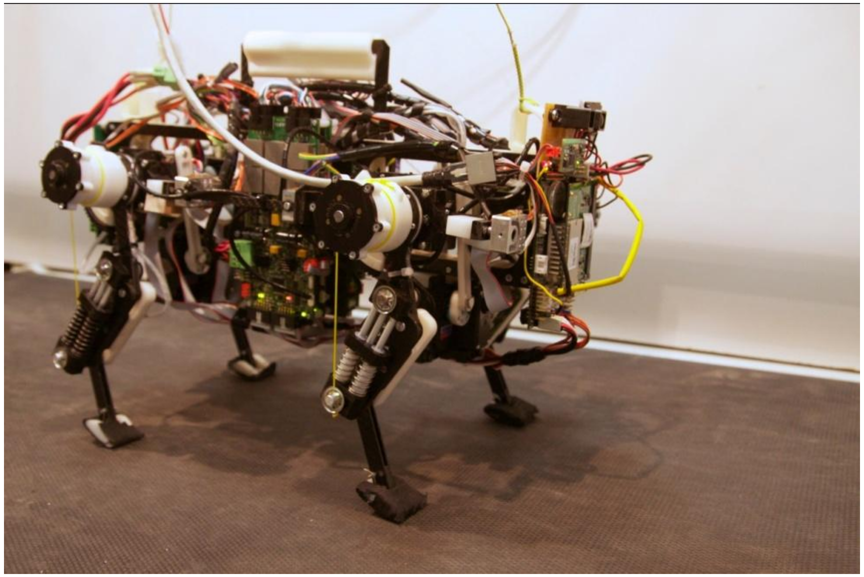
`http://benanne.github.io`

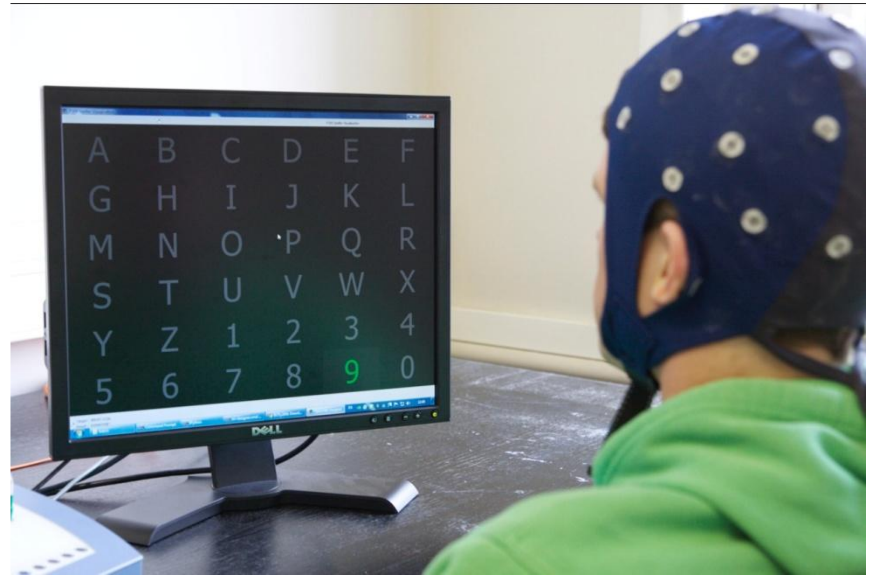
`http://github.com/benanne`

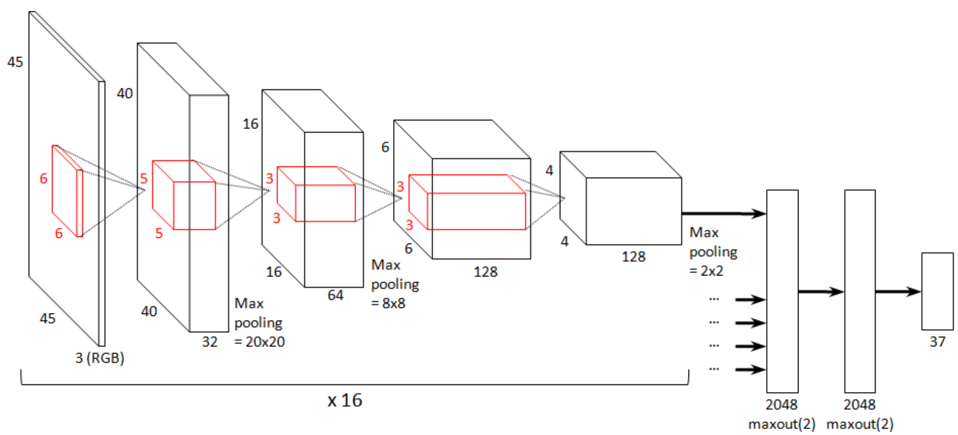
`http://reslab.elis.ugent.be`

`sanderdieleman@gmail.com`

Reservoir Lab







Code and slides:

http://github.com/benanne/theano-tutorial

CIFAR-10 Dataset:

http://www.cs.toronto.edu/~kriz/cifar.html

Based on a tutorial by **Alec Radford**:

https://github.com/newmu/Theano-Tutorials

0. Introduction

Machine learning

$\mathbf{x}_n$	training examples
$\mathbf{t}_n$	training labels

Machine learning

$\mathbf{x}_n$	training examples
$\mathbf{t}_n$	training labels
$f_\theta(\mathbf{x})$	model
θ	parameters
$\mathbf{y}_n = f_\theta(\mathbf{x}_n)$	predictions

Machine learning

$\mathbf{x}_n$	training examples
$\mathbf{t}_n$	training labels
$f_\theta(\mathbf{x})$	model
θ	parameters
$\mathbf{y}_n = f_\theta(\mathbf{x}_n)$	predictions

learning: adapt θ so that $\mathbf{y}_n \approx \mathbf{t}_n$

Machine learning

$\mathbf{x}_n$	training examples
$\mathbf{t}_n$	training labels
$f_\theta(\mathbf{x})$	model
θ	parameters
$\mathbf{y}_n = f_\theta(\mathbf{x}_n)$	predictions

learning: adapt θ so that $\mathbf{y}_n \approx \mathbf{t}_n$

+ generalization to new examples!

‘Shallow’ learning

$\mathbf{x}_n$ → $\boldsymbol{\phi}_n$ → $f_\theta(\boldsymbol{\phi}_n)$ → $\mathbf{y}_n$

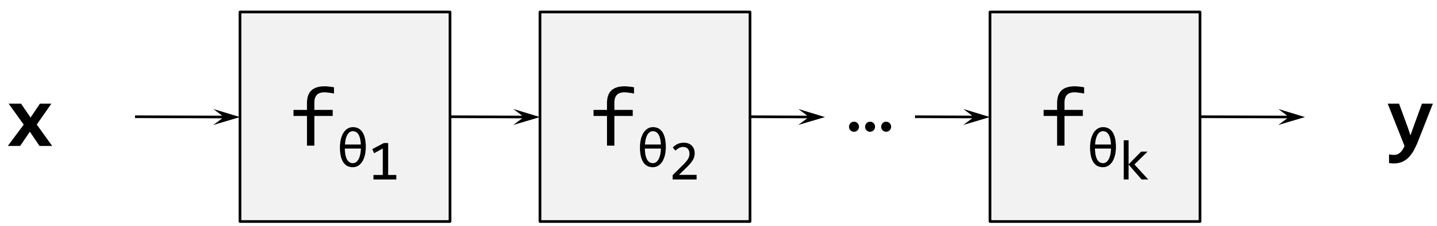
training examples → extracted features → shallow model → predictions

Deep learning

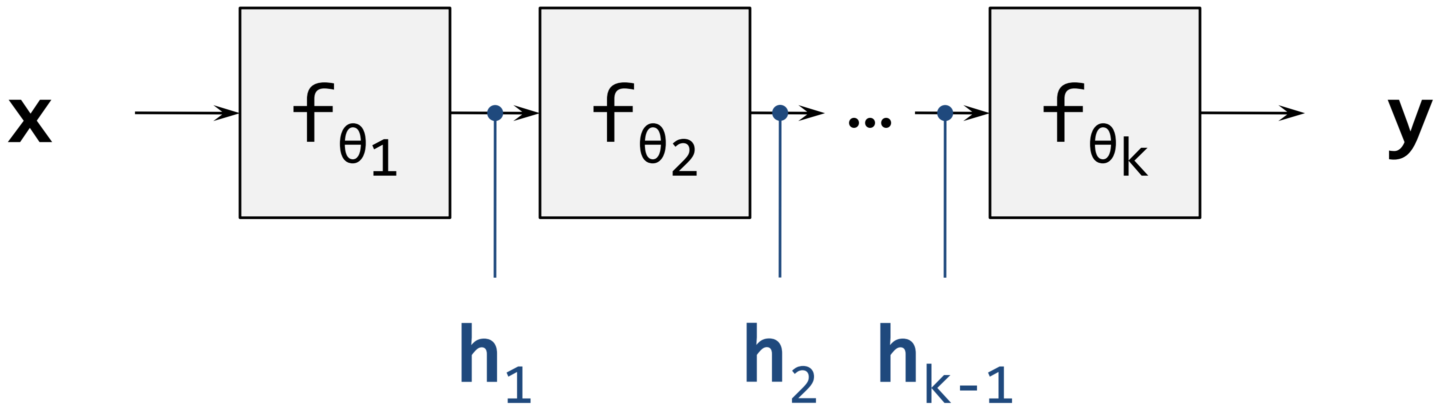
$$\mathbf{x}_n \Rightarrow f_{\theta_k}(\ldots f_{\theta_2}(f_{\theta_1}(\mathbf{x}_n))) \Rightarrow \mathbf{y}_n$$

training examples | deep model | predictions

Deep learning

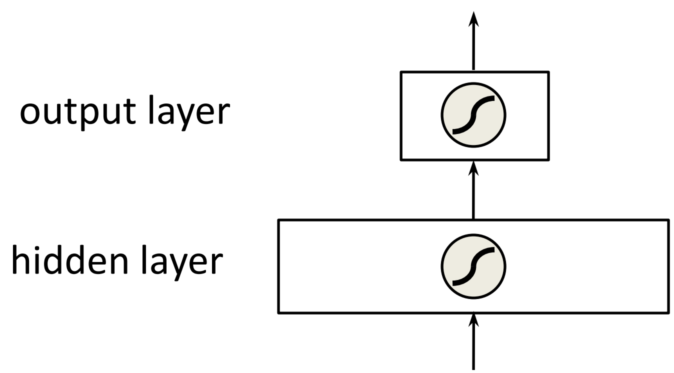


Deep learning

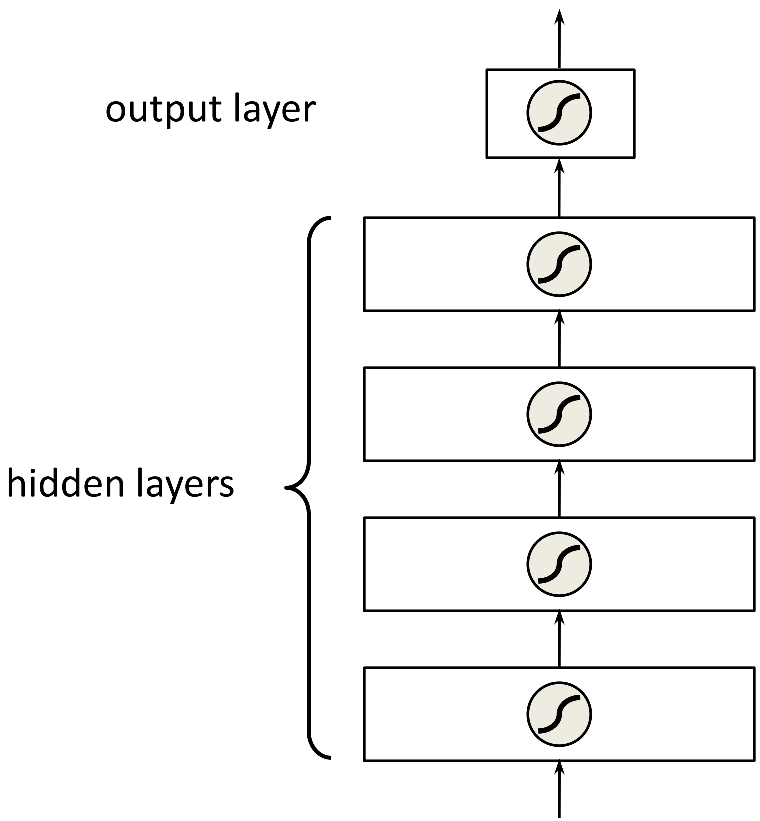


learned intermediate representations

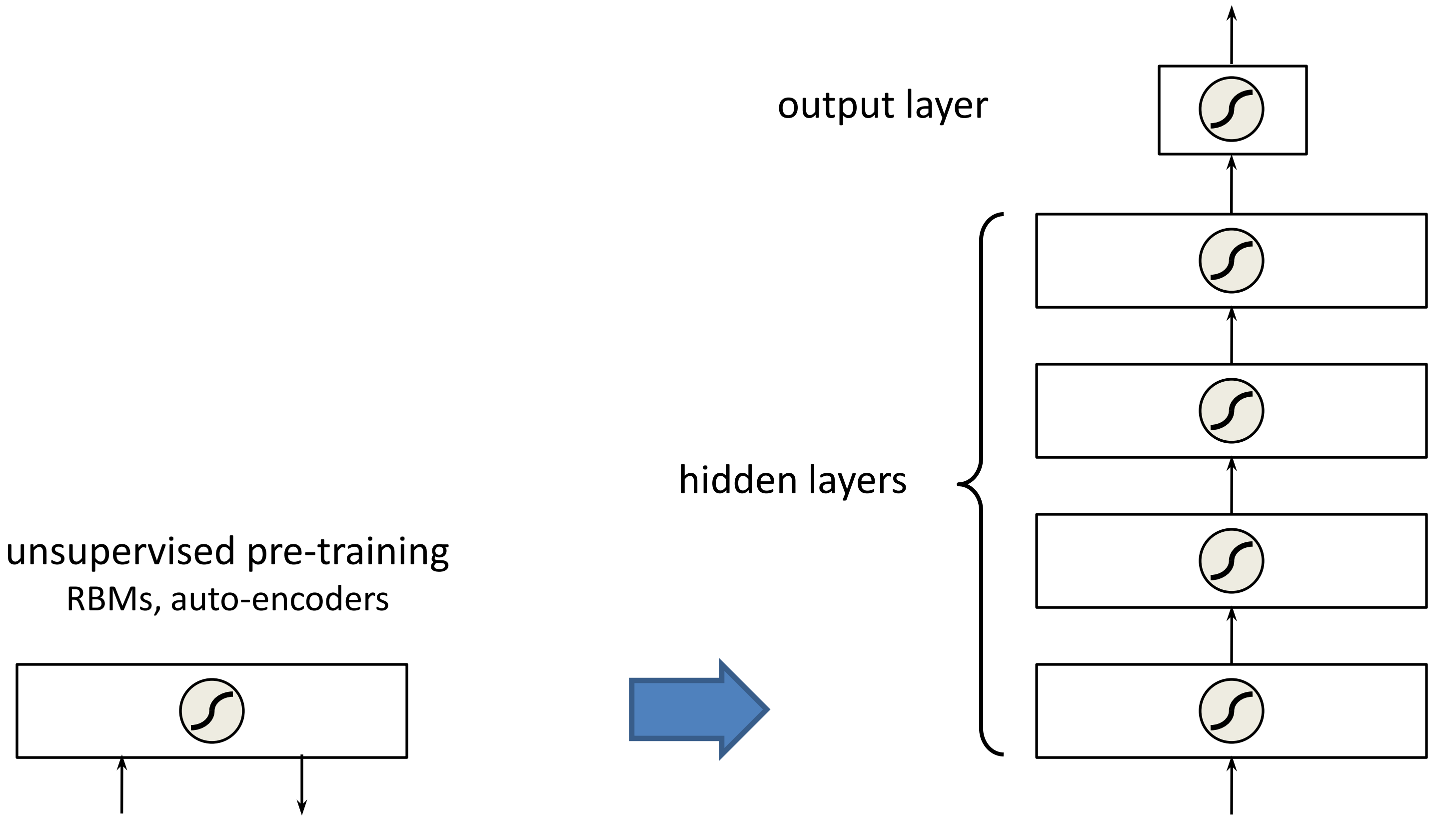
Deep learning vs. traditional neural networks



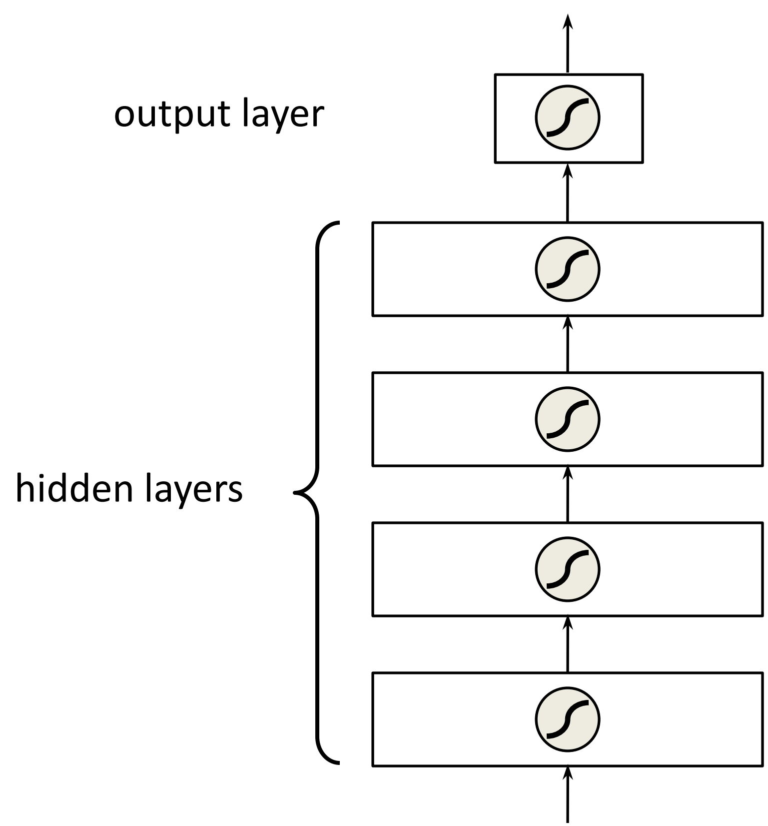
Deep learning vs. traditional neural networks



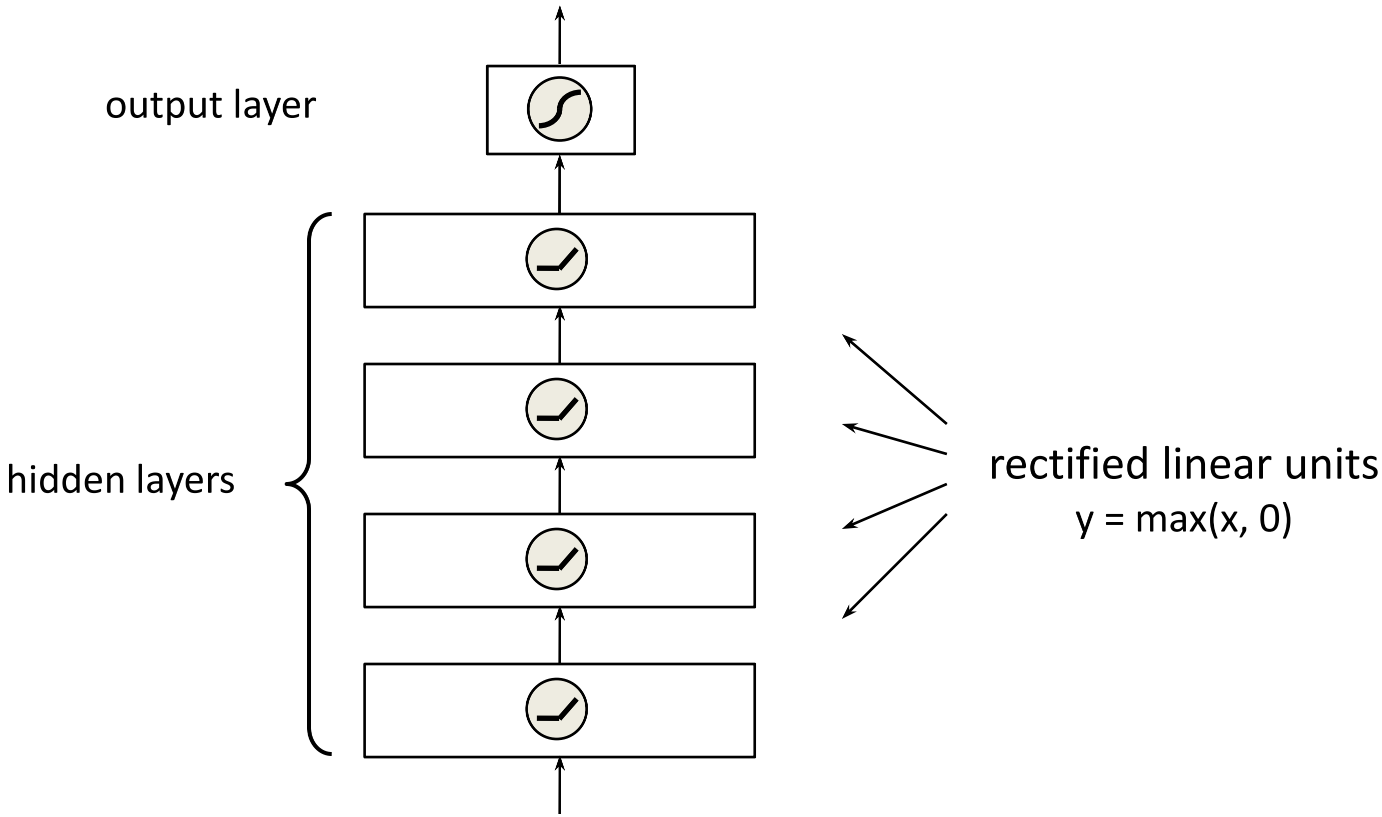
Deep learning vs. traditional neural networks



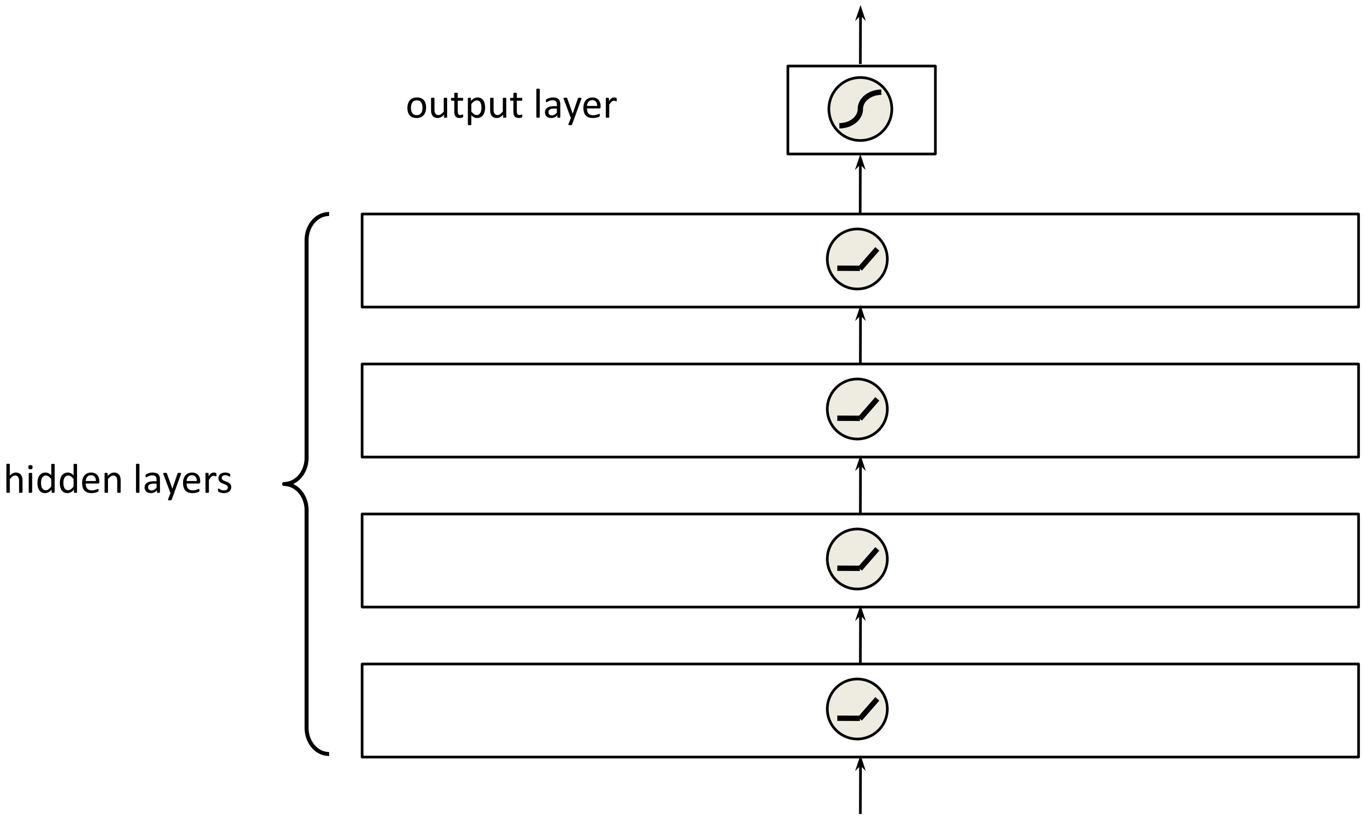
Deep learning vs. traditional neural networks



Deep learning vs. traditional neural networks



Deep learning vs. traditional neural networks



A brief history of deep learning

- 2006 — unsupervised pre-training with RBMs
- 2007 — unsupervised pre-training with auto-encoders
- 2010 — rectified linear units (ReLUs)
- 2012 — dropout regularization
- 2012 — Alex Krizhevsky wins ImageNet by a landslide

theano

- Library for python™
- Theano compiles **mathematical expressions**
- Designed with **machine learning** in mind (but also useful for other things)
- Transparent switching between **CPU** and **GPU**
- Tight integration with **NumPy**
- Popular tool for **deep learning** research (alternatives include **Torch7**, **Caffe**)

Theano knows maths:

- matrix algebra
- expression simplification / stabilization
- gradient computation
- efficient (parallelized) execution

Overview

1. Theano basics
2. Linear regression
3. Logistic regression
4. Neural network
5. Neural network (continued)
6. Convolutional neural network
7. Advanced topics

1. Theano basics

Multiplying two numbers in Python

```
>>> a = 2
>>> b = 3
>>> a * b
6
>>>
```

Multiplying two numbers in Theano (1)

```
>>> import theano
>>> import theano.tensor as T
```

Multiplying two numbers in Theano (2)

```
>>> import theano
>>> import theano.tensor as T
>>> a = T.scalar() # symbolic variables
>>> b = T.scalar()
```

Multiplying two numbers in Theano (3)

```
>>> import theano
>>> import theano.tensor as T
>>> a = T.scalar()
>>> b = T.scalar()
>>> a * b
Elemwise{mul,no_inplace}.0
```

Multiplying two numbers in Theano (4)

```
>>> import theano
>>> import theano.tensor as T
>>> a = T.scalar()
>>> b = T.scalar()
>>> y = a * b
>>> theano.function([a, b], y)
<theano.compile.function_module.Function object
  at 0x7e4dad0>
```

Multiplying two numbers in Theano (5)

```
>>> import theano
>>> import theano.tensor as T
>>> a = T.scalar()
>>> b = T.scalar()
>>> y = a * b
>>> f = theano.function([a, b], y)
>>> f(1, 2)
array(2.0)
>>> f(3, 3)
array(9.0)
```

Symbolic and numerical computation

```
>>> a = T.scalar()
>>> b = T.scalar()
>>> y = a * b
```

Symbolic computation

```
>>> f = theano.function([a, b], y)
```

```
>>> f(1, 2)
array(2.0)
>>> f(3, 3)
array(9.0)
```

Numerical computation

2. Linear regression

Linear regression (1)

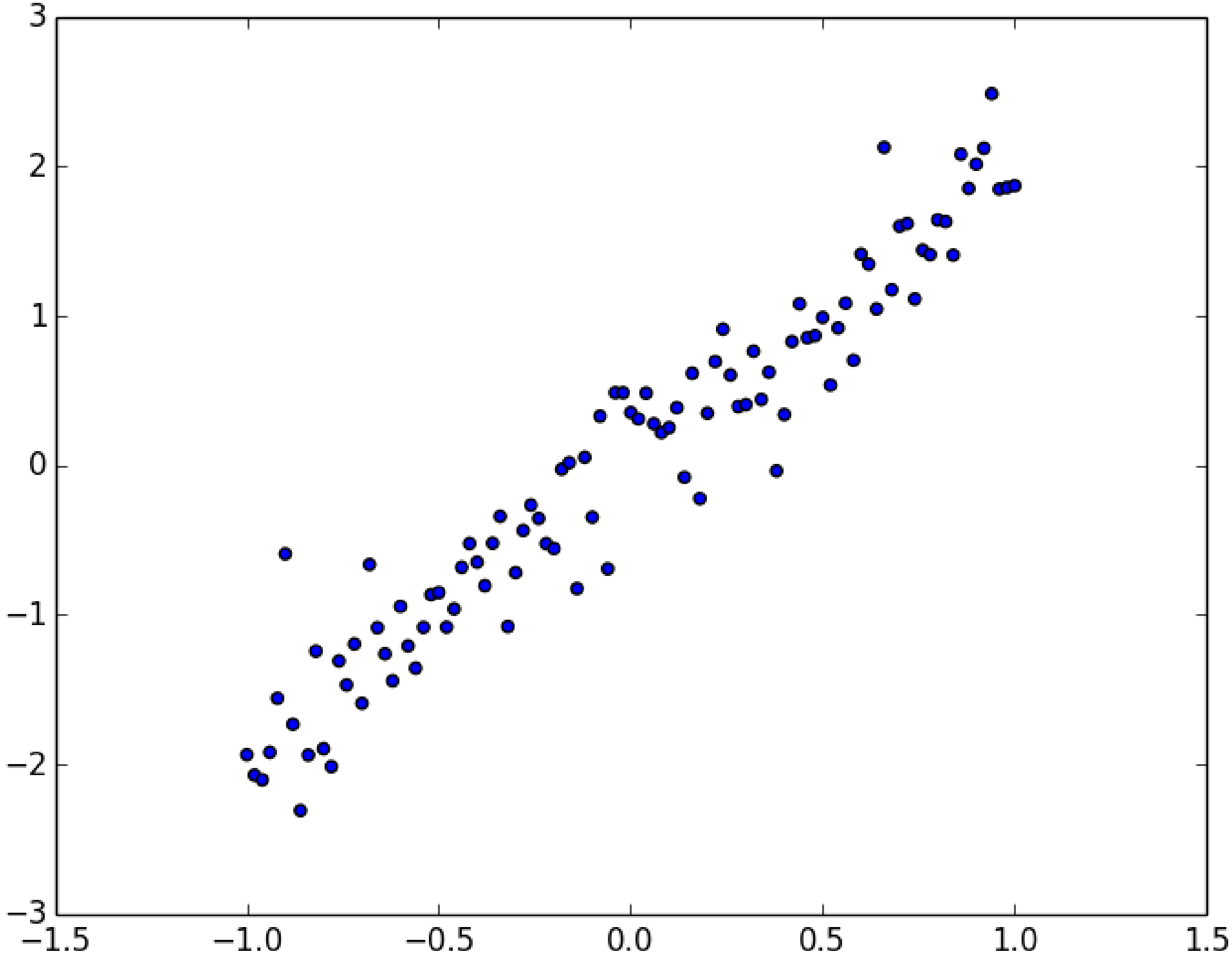
```
import theano
import theano.tensor as T
import numpy as np
import matplotlib.pyplot as plt
plt.ion()
```

Linear regression (2)

```
import theano
import theano.tensor as T
import numpy as np
import matplotlib.pyplot as plt
plt.ion()

# create artificial training data
x_train = np.linspace(-1, 1, 101)
t_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.33
```

Linear regression (3)

```
import theano
import theano.tensor as T
import numpy as np
import matplotlib.pyplot as plt
plt.ion()

# create artificial training data
x_train = np.linspace(-1, 1, 101)
t_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.33

# plot data
plt.scatter(x_train, t_train)
```



3
2
1
0
−1
−2
−3
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

Linear regression (4)

```
…
# create artificial training data
x_train = np.linspace(-1, 1, 101)
t_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.33

# plot data
plt.scatter(x_train, t_train)

# define symbolic Theano variables
x = T.scalar()
t = T.scalar()
```

Linear regression (5)

```
…
# define symbolic Theano variables
x = T.scalar()
t = T.scalar()

# define model: linear regression
def model(x, w):
    return x * w
```

Linear regression (6)

```
…
# define symbolic Theano variables
x = T.scalar()
t = T.scalar()

# define model: linear regression
def model(x, w):
    return x * w

w = theano.shared(0.0)
y = model(x, w)
```

Linear regression (7)

```
…
# define symbolic Theano variables
x = T.scalar()
t = T.scalar()

# define model: linear regression
def model(x, w):
    return x * w

w = theano.shared(0.0)
y = model(x, w)

cost = T.mean((t - y) ** 2)
```

Linear regression (8)

```
…
# define model: linear regression
def model(x, w):
    return x * w

w = theano.shared(0.0)
y = model(x, w)

cost = T.mean((t - y) ** 2)

g = T.grad(cost, w) # magic!
updates = [(w, w - g * 0.01)]
```

Linear regression (9)

```
…
cost = T.mean((t - y) ** 2)

g = T.grad(cost, w)
updates = [(w, w - g * 0.01)]

# compile Theano function
train = theano.function([x, t], cost, updates=updates)
```

Linear regression (10)

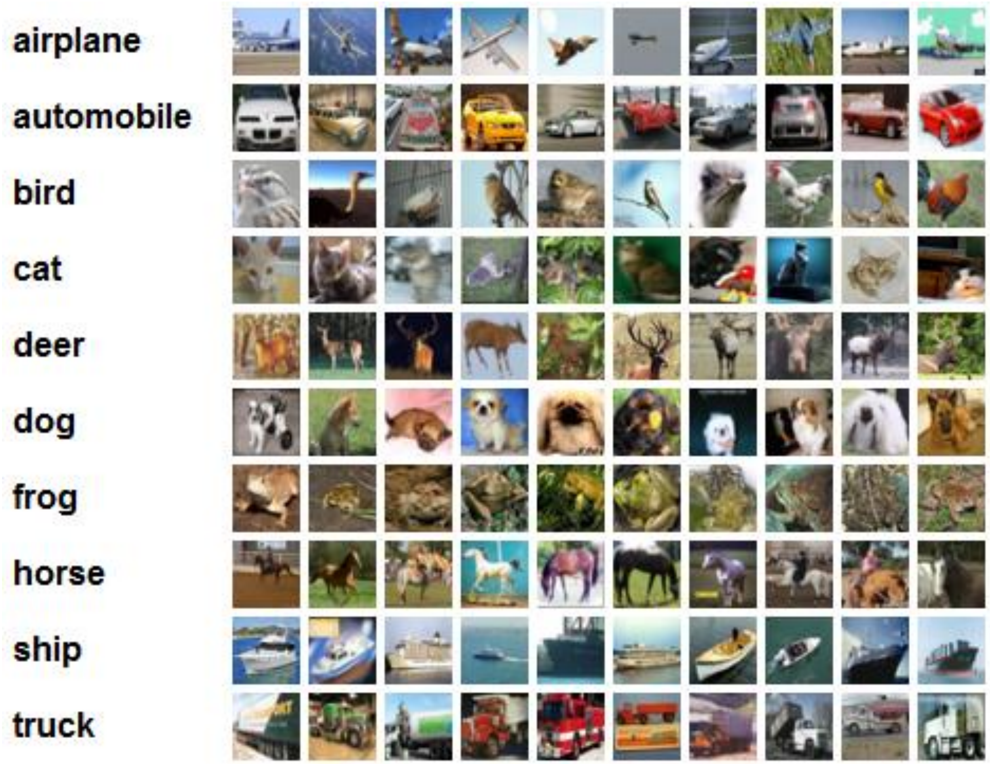
```
…
# compile Theano function
train = theano.function([x, t], cost, updates=updates)

# train model
for i in range(20):
    print "iteration %d" % (i + 1)
    for x, t in zip(x_train, t_train):
        train(x, t)

    print "w = %.8f" % w.get_value()
    print
```

Linear regression (11)

```
…
# train model
for i in range(20):
    print "iteration %d" % (i + 1)
    for x, t in zip(x_train, t_train):
        train(x, t)

    print "w = %.8f" % w.get_value()
    print

# plot fitted line
plt.plot(x_train, w.get_value() * x_train)
```

3. Logistic regression

The CIFAR-10 dataset



t: labels **x**: 32 by 32 pixel color images

Logistic regression (1)

$$p(y = k \mid x) = \frac{\exp(w_k x)}{\sum_l \exp(w_l x)}$$

$$\arg\max_{w_l} \sum_n p(y_n = t_n \mid x_n)$$

Logistic regression (2)

```
import theano
import theano.tensor as T
import numpy as np
import matplotlib.pyplot as plt
plt.ion()

import load
```

Logistic regression (3)

```
import theano
import theano.tensor as T
import numpy as np
import matplotlib.pyplot as plt
plt.ion()

import load

# load data
x_train, t_train, x_test, t_test =
   load.cifar10(dtype=theano.config.floatX)

labels_test = np.argmax(t_test, axis=1)
```

Logistic regression (4)

```
…
# load data
x_train, t_train, x_test, t_test =
   load.cifar10(dtype=theano.config.floatX)

labels_test = np.argmax(t_test, axis=1)

# visualize data
plt.imshow(x_train[0].reshape(32, 32), cmap=plt.cm.gray)
```

Logistic regression (5)

```
…
# load data
x_train, t_train, x_test, t_test =
   load.cifar10(dtype=theano.config.floatX)

labels_test = np.argmax(t_test, axis=1)

# visualize data
plt.imshow(x_train[0].reshape(32, 32), cmap=plt.cm.gray)

# define symbolic Theano variables
x = T.matrix()
t = T.matrix()
```

Logistic regression (6)

```
…
# define symbolic Theano variables
x = T.matrix()
t = T.matrix()

# define model: logistic regression
def floatX(x):
    return np.asarray(x, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.1))
```

Logistic regression (7)

```
…
# define model: logistic regression
def floatX(x):
    return np.asarray(x, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.1))

def model(x, w):
    return T.nnet.softmax(T.dot(x, w))
```

Logistic regression (8)

```
…
def model(x, w):
    return T.nnet.softmax(T.dot(x, w))

w = init_weights((32 * 32, 10))

p_y_given_x = model(x, w)
y = T.argmax(p_y_given_x, axis=1)
```

Logistic regression (9)

```
…
def model(x, w):
    return T.nnet.softmax(T.dot(x, w))

w = init_weights((32 * 32, 10))

p_y_given_x = model(x, w)
y = T.argmax(p_y_given_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(p_y_given_x, t))
g = T.grad(cost, w)
updates = [(w, w - g * 0.001)]
```

Logistic regression (10)

```
…
cost = T.mean(T.nnet.categorical_crossentropy(p_y_given_x, t))
g = T.grad(cost, w)
updates = [(w, w - g * 0.001)]

# compile theano functions
train = theano.function([x, t], cost, updates=updates)
predict = theano.function([x], y)
```

Logistic regression (10)

```
…
# train model
batch_size = 50
for i in range(100):
    print "iteration %d" % (i + 1)
    for start in range(0, len(x_train), batch_size):
        x_batch = x_train[start:start + batch_size]
        t_batch = t_train[start:start + batch_size]
        cost = train(x_batch, t_batch)

    predictions_test = predict(x_test)
    accuracy = np.mean(predictions_test == labels_test)
    print "accuracy: %.5f" % accuracy
    print
```

4. Neural network

Neural network (1)

```
…
# define model: neural network
def floatX(x):
    return np.asarray(x, dtype=theano.config.floatX)

def init_weights(shape):
    return theano.shared(floatX(np.random.randn(*shape) * 0.1))

def sgd(cost, params, learning_rate):
    grads = T.grad(cost, params)
    updates = []
    for p, g in zip(params, grads):
        updates.append([p, p - g * learning_rate])
    return updates
```

Neural network (2)

```
…
def sgd(cost, params, learning_rate):
    grads = T.grad(cost, params)
    updates = []
    for p, g in zip(params, grads):
        updates.append([p, p - g * learning_rate])
    return updates

def model(x, w_h, w_o):
    h = T.maximum(0, T.dot(x, w_h))
    p_y_given_x = T.nnet.softmax(T.dot(h, w_o))
    return p_y_given_x
```

Neural network (3)

```
…
def model(x, w_h, w_o):
    h = T.maximum(0, T.dot(x, w_h))
    p_y_given_x = T.nnet.softmax(T.dot(h, w_o))
    return p_y_given_x

w_h = init_weights((32 * 32, 100))
w_o = init_weights((100, 10))

p_y_given_x = model(x, w_h, w_o)
y = T.argmax(p_y_given_x, axis=1)
```

Neural network (3)

```
…
w_h = init_weights((32 * 32, 100))
w_o = init_weights((100, 10))

p_y_given_x = model(x, w_h, w_o)
y = T.argmax(p_y_given_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(p_y_given_x, t))
params = [w_h, w_o]
updates = sgd(cost, params, learning_rate=0.01)
```

Neural network (4)

```
…
cost = T.mean(T.nnet.categorical_crossentropy(p_y_given_x, t))
params = [w_h, w_o]
updates = sgd(cost, params, learning_rate=0.01)

# compile theano functions
train = theano.function([x, t], cost, updates=updates)
predict = theano.function([x], y)
```

Neural network (5)

```
…
# train model
batch_size = 50
for i in range(50):
    print "iteration %d" % (i + 1)
    for start in range(0, len(x_train), batch_size):
        x_batch = x_train[start:start + batch_size]
        t_batch = t_train[start:start + batch_size]
        cost = train(x_batch, t_batch)

    predictions_test = predict(x_test)
    accuracy = np.mean(predictions_test == labels_test)
    print "accuracy: %.5f" % accuracy
    print
```

5. Neural network (continued)

Exercises

- Change the learning algorithm from SGD to momentum

- Add biases

- Add an extra layer

Momentum

```
def momentum(cost, params, learning_rate, momentum):
    grads = theano.grad(cost, params)
    updates = []

    for p, g in zip(params, grads):
        mparam_i = theano.shared(np.zeros(p.get_value().shape,
   dtype=theano.config.floatX))
        v = momentum * mparam_i - learning_rate * g
        updates.append((mparam_i, v))
        updates.append((p, p + v))

    return updates
```

6. Convolutional neural network

Convolution = linear filtering

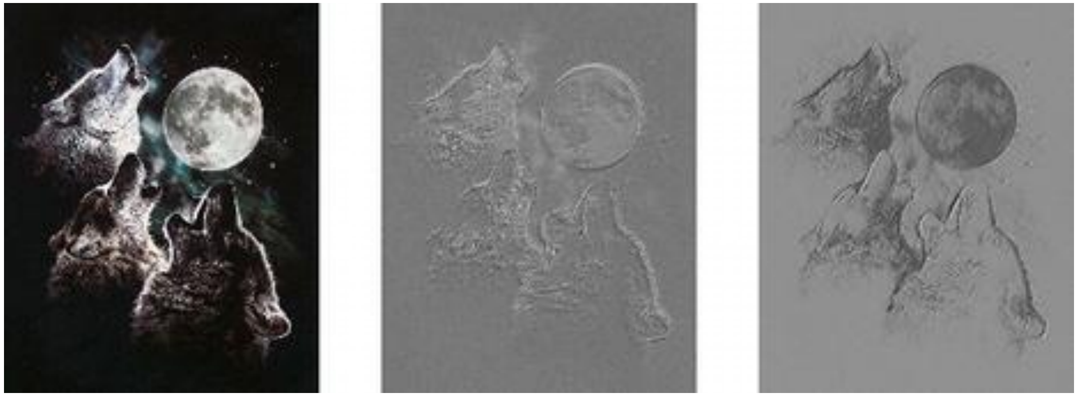


Figure reproduced from http://www.deeplearning.net/tutorial/lenet.html

Convnets are neural nets with some dot products replaced by convolutions

… + subsampling

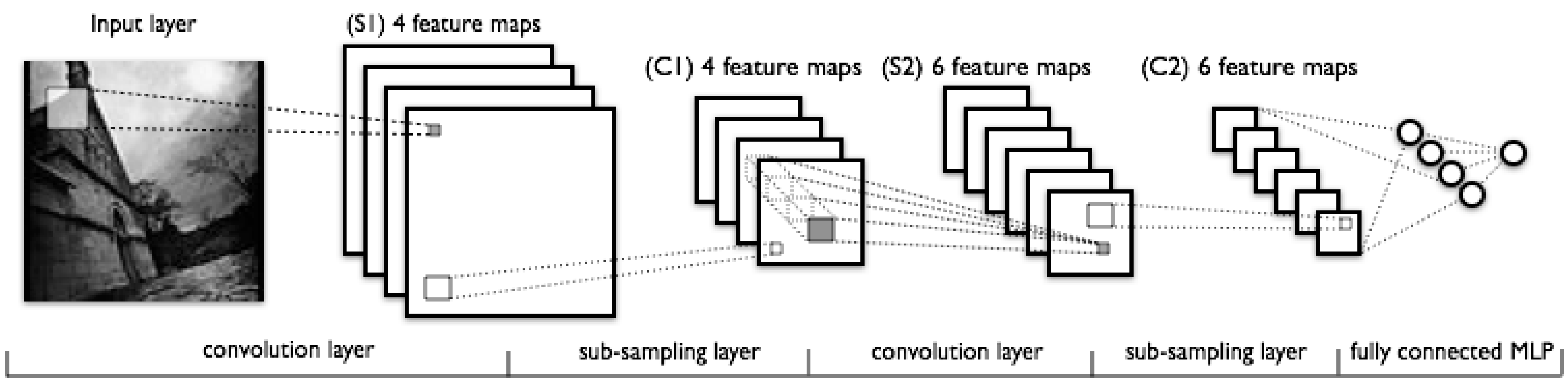
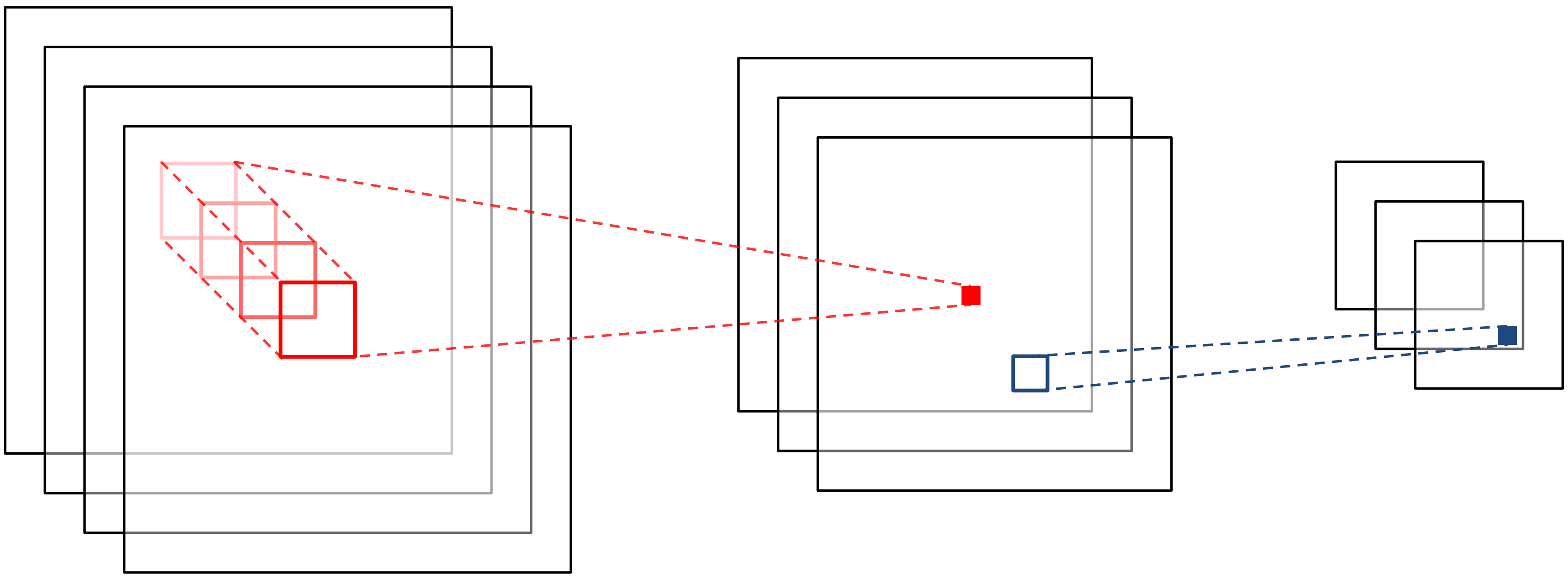


Figure reproduced from http://www.deeplearning.net/tutorial/lenet.html

A conv. layer computes a sum of convolutions with input feature maps



… a subsampling or ‘pooling’ layer computes some aggregation function of small local regions of feature maps (typically the maximum).

Note: using GPU acceleration

```
# use the CPU (default)
python script.py

# use the CPU (explicit)
THEANO_FLAGS=device=cpu python script.py

# use the GPU (only single precision is supported)
THEANO_FLAGS=device=gpu,floatX=float32 python script.py

# … or create a .theanorc file in your home directory:
[global]
floatX = float32
device = gpu
```

Convolutional neural network (1)

```
import theano
import theano.tensor as T
import numpy as np
import matplotlib.pyplot as plt
plt.ion()

import load

from theano.tensor.nnet.conv import conv2d
from theano.tensor.signal.downsample import max_pool_2d
```

Convolutional neural network (2)

```
…
# load data
x_train, t_train, x_test, t_test =
   load.cifar10(dtype=theano.config.floatX)
labels_test = np.argmax(t_test, axis=1)

# reshape data
x_train = x_train.reshape((x_train.shape[0], 1, 32, 32))
x_test = x_test.reshape((x_test.shape[0], 1, 32, 32))
```

Convolutional neural network (3)

```
…
# load data
x_train, t_train, x_test, t_test =
   load.cifar10(dtype=theano.config.floatX)
labels_test = np.argmax(t_test, axis=1)

# reshape data
x_train = x_train.reshape((x_train.shape[0], 1, 32, 32))
x_test = x_test.reshape((x_test.shape[0], 1, 32, 32))

# define symbolic Theano variables
x = T.tensor4()
t = T.matrix()
```

Convolutional neural network (4)

```
…
def model(x, w_c1, b_c1, w_c2, b_c2, w_h3, b_h3, w_o, b_o):
    c1 = T.maximum(0, conv2d(x, w_c1) + b_c1.dimshuffle('x', 0,
   'x', 'x'))
    p1 = max_pool_2d(c1, (3, 3))
```

Convolutional neural network (5)

```
…
def model(x, w_c1, b_c1, w_c2, b_c2, w_h3, b_h3, w_o, b_o):
    c1 = T.maximum(0, conv2d(x, w_c1) + b_c1.dimshuffle('x', 0, 'x', 'x'))
    p1 = max_pool_2d(c1, (3, 3))

    c2 = T.maximum(0, conv2d(p1, w_c2) + b_c2.dimshuffle('x', 0, 'x', 'x'))
    p2 = max_pool_2d(c2, (2, 2))
```

Convolutional neural network (6)

```
…
def model(x, w_c1, b_c1, w_c2, b_c2, w_h3, b_h3, w_o, b_o):
    c1 = T.maximum(0, conv2d(x, w_c1) + b_c1.dimshuffle('x', 0,
   'x', 'x'))
    p1 = max_pool_2d(c1, (3, 3))

    c2 = T.maximum(0, conv2d(p1, w_c2) + b_c2.dimshuffle('x', 0,
   'x', 'x'))
    p2 = max_pool_2d(c2, (2, 2))

    p2_flat = p2.flatten(2)
    h3 = T.maximum(0, T.dot(p2_flat, w_h3) + b_h3)
    p_y_given_x = T.nnet.softmax(T.dot(h3, w_o) + b_o)
    return p_y_given_x
```

Convolutional neural network (7)

```
…
w_c1 = init_weights((4, 1, 3, 3))
b_c1 = init_weights((4,))
w_c2 = init_weights((8, 4, 3, 3))
b_c2 = init_weights((8,))
w_h3 = init_weights((8 * 4 * 4, 100))
b_h3 = init_weights((100,))
w_o = init_weights((100, 10))
b_o = init_weights((10,))

params = [w_c1, b_c1, w_c2, b_c2, w_h3, b_h3, w_o, b_o]

p_y_given_x = model(x, *params)
y = T.argmax(p_y_given_x, axis=1)
```

Convolutional neural network (8)

```
…
p_y_given_x = model(x, *params)
y = T.argmax(p_y_given_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(p_y_given_x, t))
updates = momentum(cost, params, learning_rate=0.01,
   momentum=0.9)
```

Convolutional neural network (9)

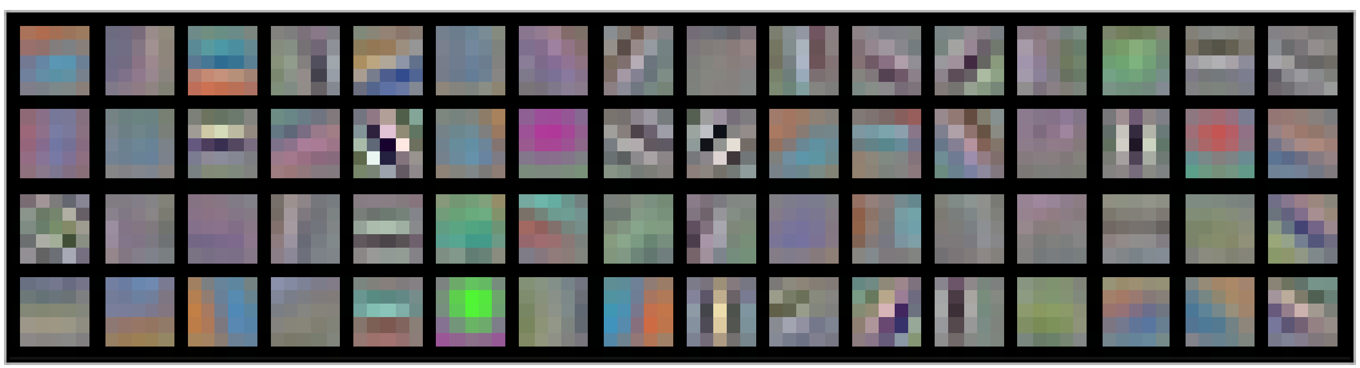
```
…
p_y_given_x = model(x, *params)
y = T.argmax(p_y_given_x, axis=1)

cost = T.mean(T.nnet.categorical_crossentropy(p_y_given_x, t))
updates = momentum(cost, params, learning_rate=0.01,
   momentum=0.9)

# compile theano functions
train = theano.function([x, t], cost, updates=updates)
predict = theano.function([x], y)
```

Convolutional neural network (10)

```
…
# train model
batch_size = 50
for i in range(50):
    print "iteration %d" % (i + 1)
    for start in range(0, len(x_train), batch_size):
        x_batch = x_train[start:start + batch_size]
        t_batch = t_train[start:start + batch_size]
        cost = train(x_batch, t_batch)

    predictions_test = predict(x_test)
    accuracy = np.mean(predictions_test == labels_test)
    print "accuracy: %.5f" % accuracy
    print
```

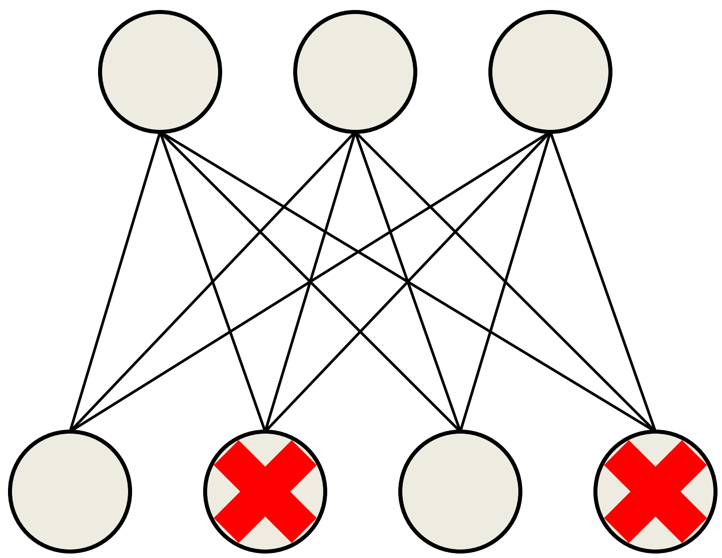
Note: visualizing filters in a grid

```
import plot_utils
plot_utils.visualize_grid(w_c1.get_value())

plot_utils.visualize_grid(w_c2.get_value()[:, 0])
plot_utils.visualize_grid(w_c2.get_value()[:, 1])
…
```



Dropout regularization (1)

Randomly remove units from the network during training to prevent **co-adaptation**



Dropout regularization (2)

```
# from theano.tensor.shared_randomstreams import RandomStreams
from theano.sandbox.rng_mrg import MRG_RandomStreams as
   RandomStreams
srng = RandomStreams()
…
def dropout(x, p=0.):
    if p > 0:
        retain_prob = 1 - p
        x *= srng.binomial(x.shape, p=retain_prob,
   dtype=theano.config.floatX)
        x /= retain_prob
    return x
```

Exercises

- Add dropout between the topmost convolutional layer and the dense hidden layer of the network

- Switch to using color images

```
x_train, t_train, x_test, t_test =
   load.cifar10(dtype=theano.config.floatX, grayscale=False)
```

7. Advanced topics

Debugging in Theano

Debugging is nontrivial: executed code was compiled and optimized!

- theano.printing.debugprint
- theano.printing.Print()
- debugmode

Model debugging

Sometimes your code is right, but your model is wrong

- compute and visualize activations, weights, updates
- … or their statistics (mean, std. dev. , rms, ...)
- visualize learned filters

Performance debugging

If your code is slower than expected, use **profile mode** to see what takes up the most time

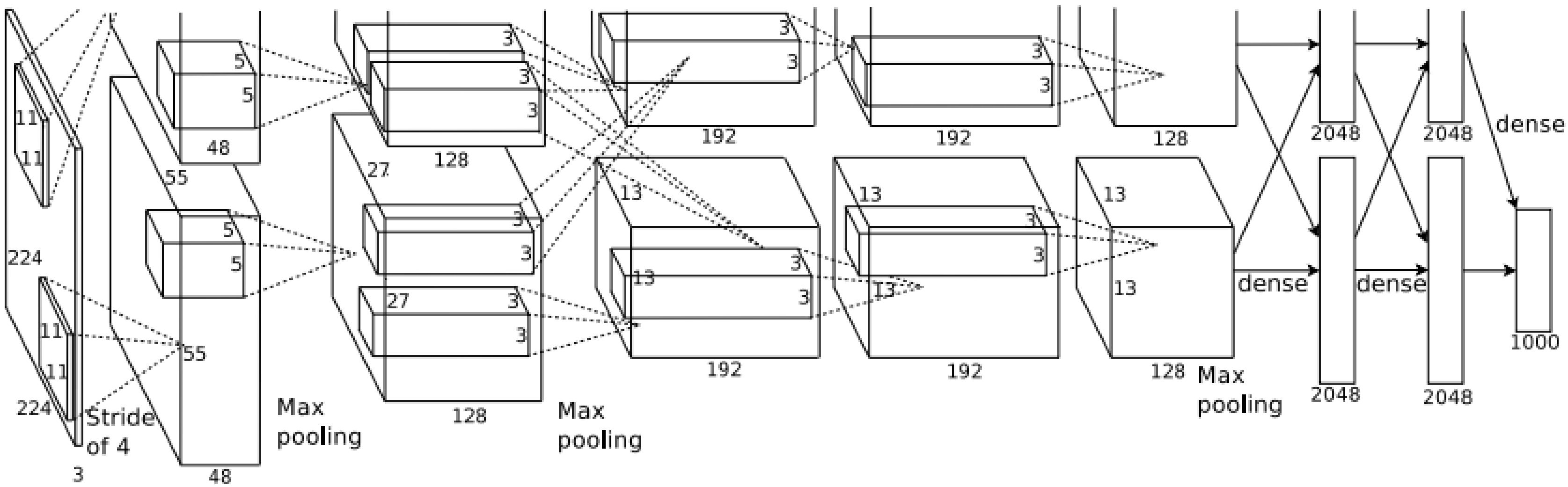
```
THEANO_FLAGS=profile=True python script.py
```

Getting help with Theano

`https://groups.google.com/forum/#!forum/theano-users`

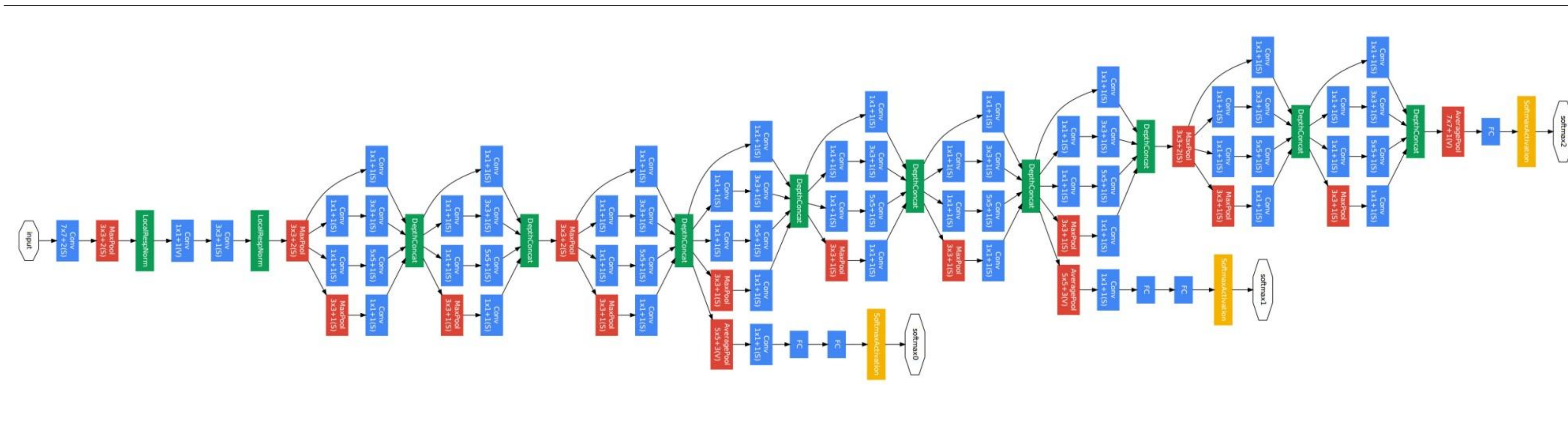
Theano has a couple of full-time developers and they are extremely helpful!

Example: AlexNet



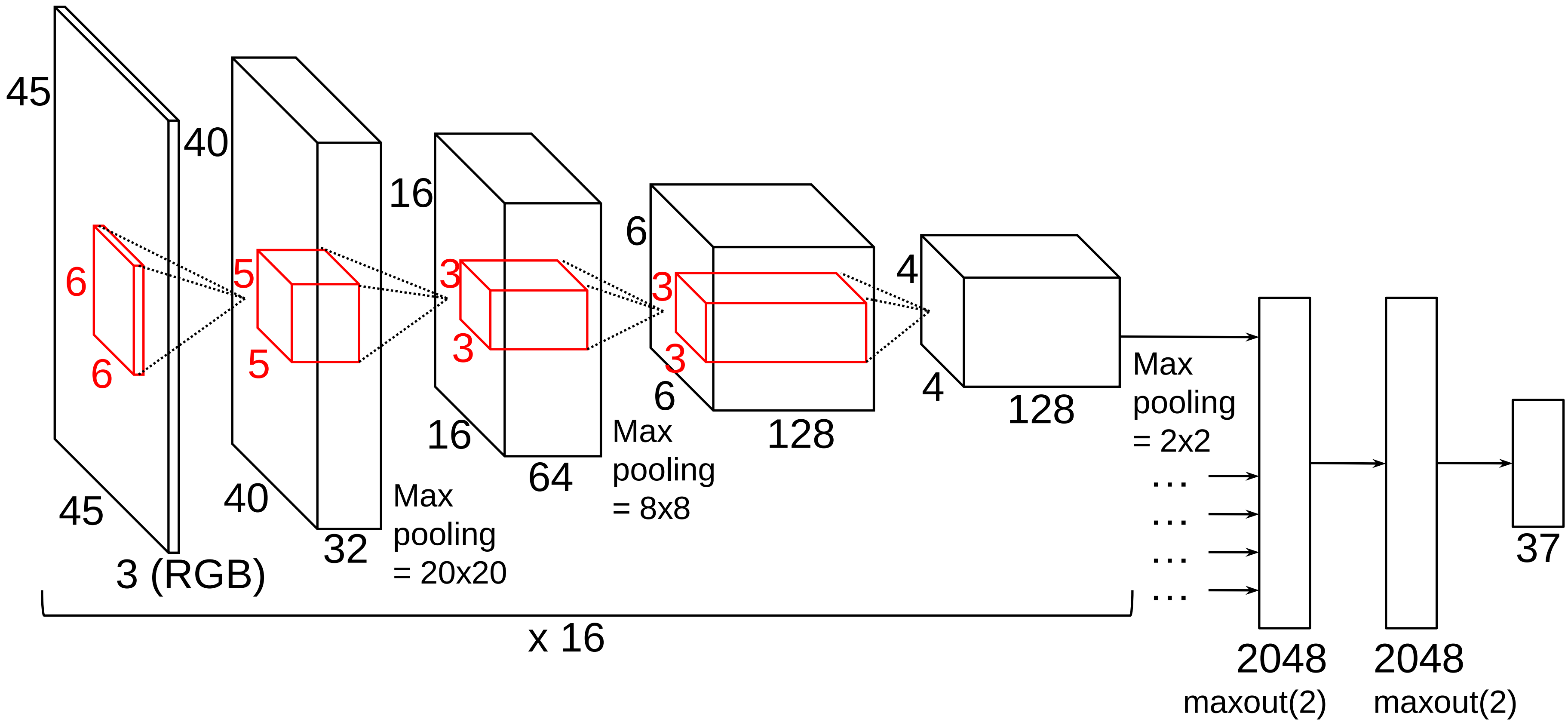
ImageNet classification with deep convolutional neural networks, Krizhevsky et al. (2012)

Example: GoogLeNet



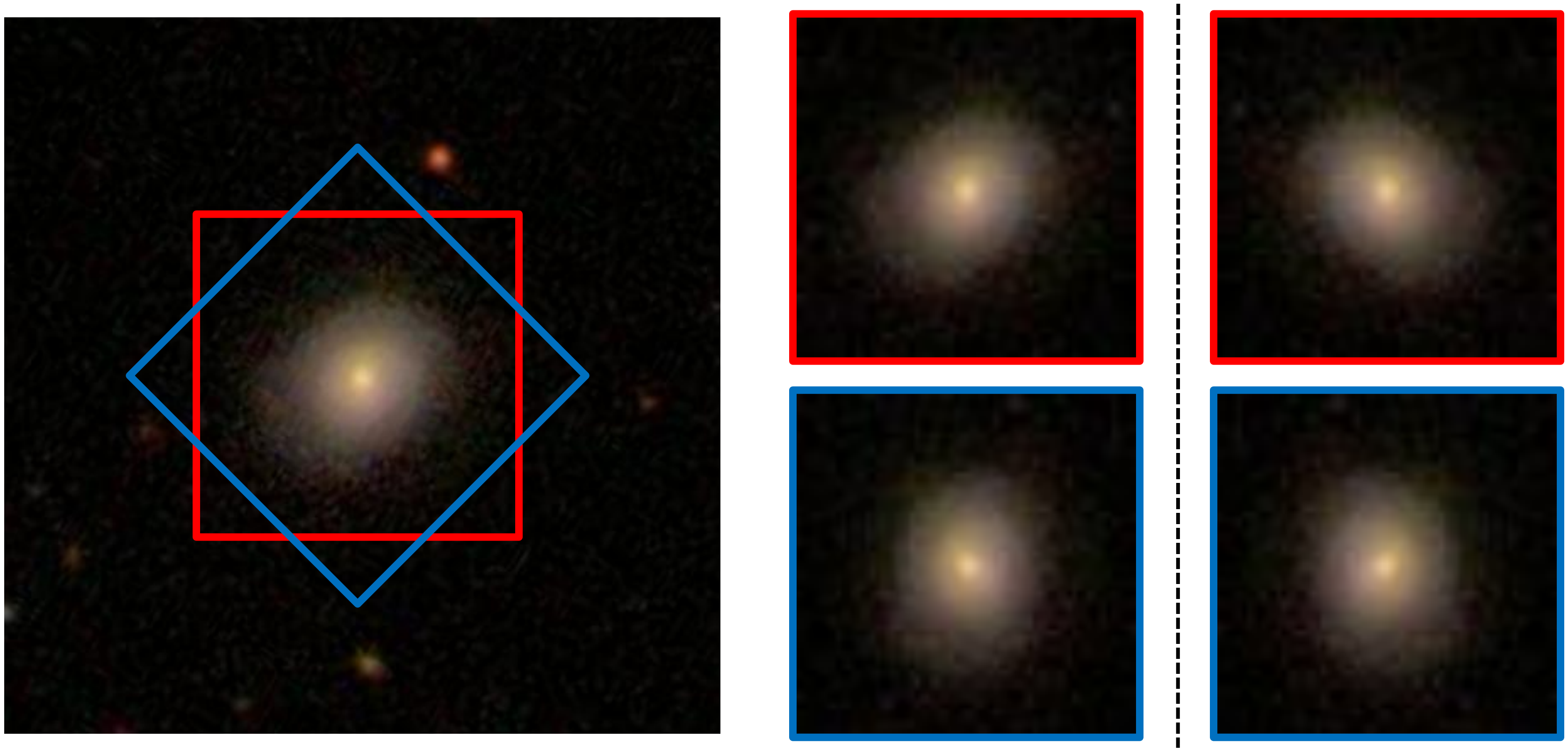
Going Deeper with Convolutions, Szegedy et al. (2014)

Example: Galaxy Challenge (1)

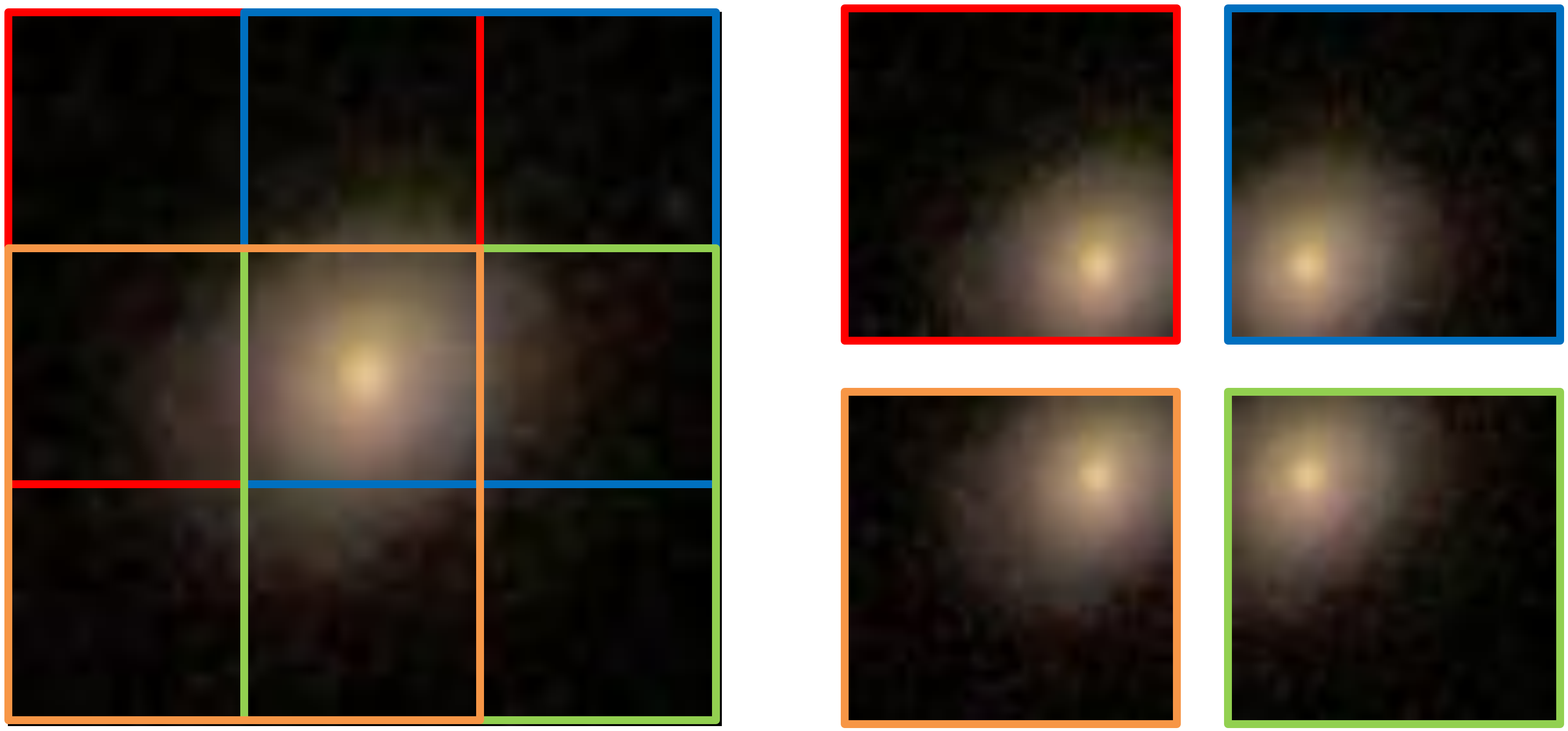


http://benanne.github.io/2014/04/05/galaxy-zoo.html

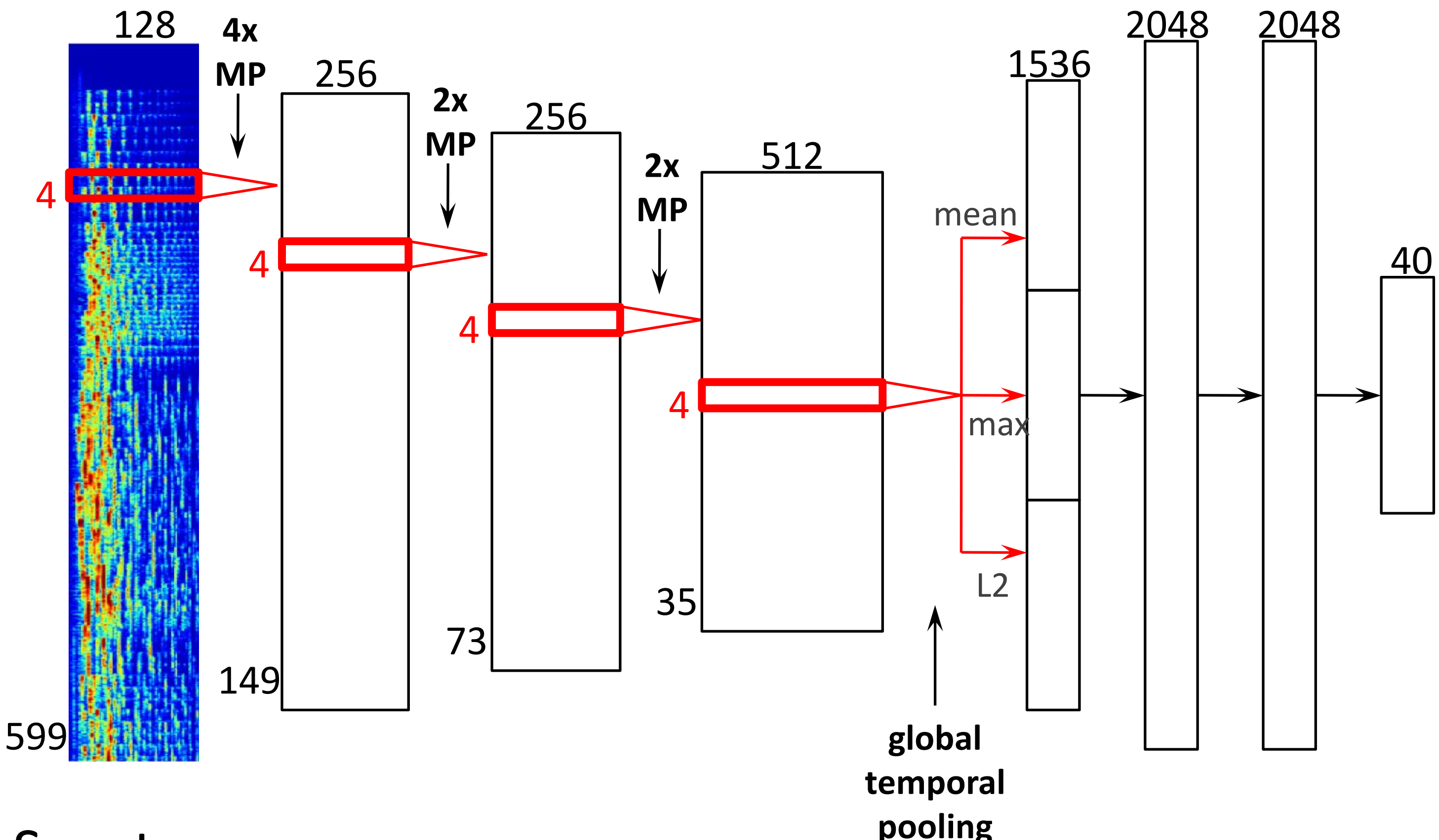
Example: Galaxy Challenge (2)



Example: Galaxy Challenge (3)



Example: content-based music recommendation at Spotify



Spectrograms

http://benanne.github.io/2014/08/05/spotify-cnns.html

Lasagne: a toolbox for building neural networks in Theano

Work in progress!

`http://github.com/benanne/Lasagne`