Deep Learning

ddebarr@uw.edu

2016-06-09

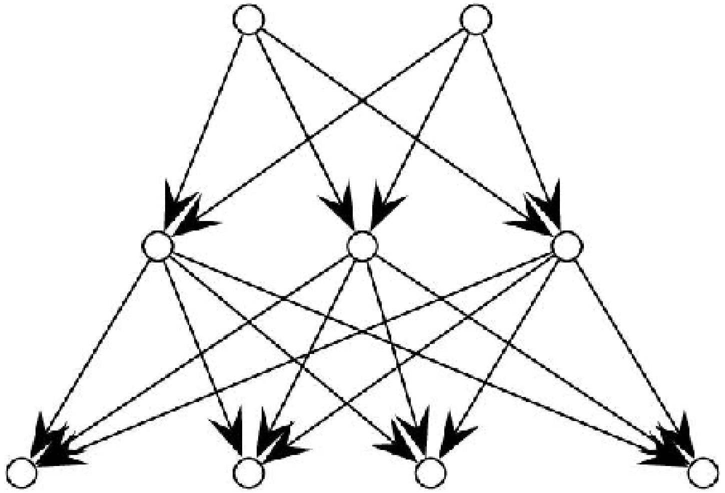
Deep Neural Networks

- More than one hidden layer
- Supervised
  - Multi-Layer Perceptron (MLP)
  - Convolutional Neural Network (CNN)
- Unsupervised
  - Auto-Encoders
  - Restricted Boltzmann Machines (RBM)

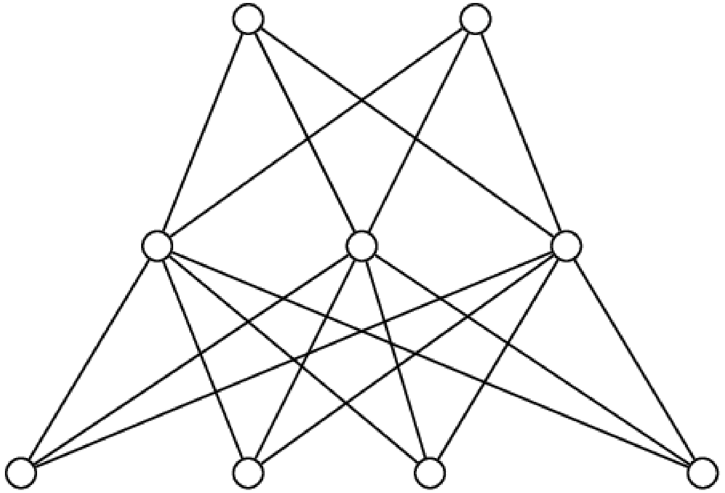
Agenda

- Deep Generative Models
- Deep Neural Nets
- Applications of Deep Neural Networks

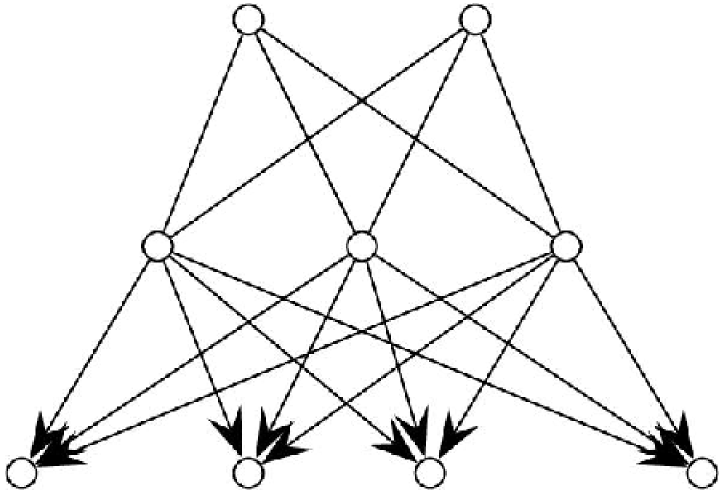
Multi-Layer Graphical Models



Deep Directed Network

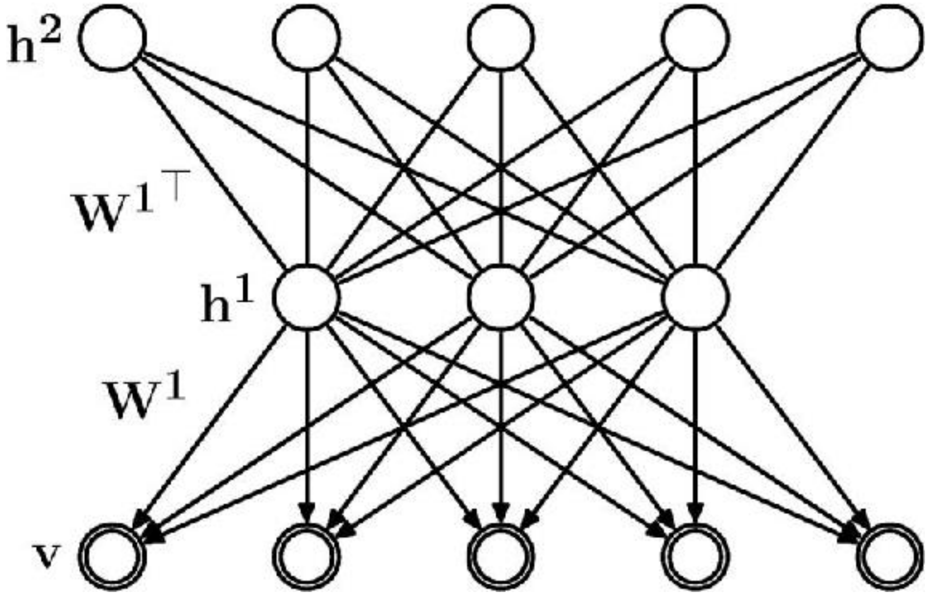


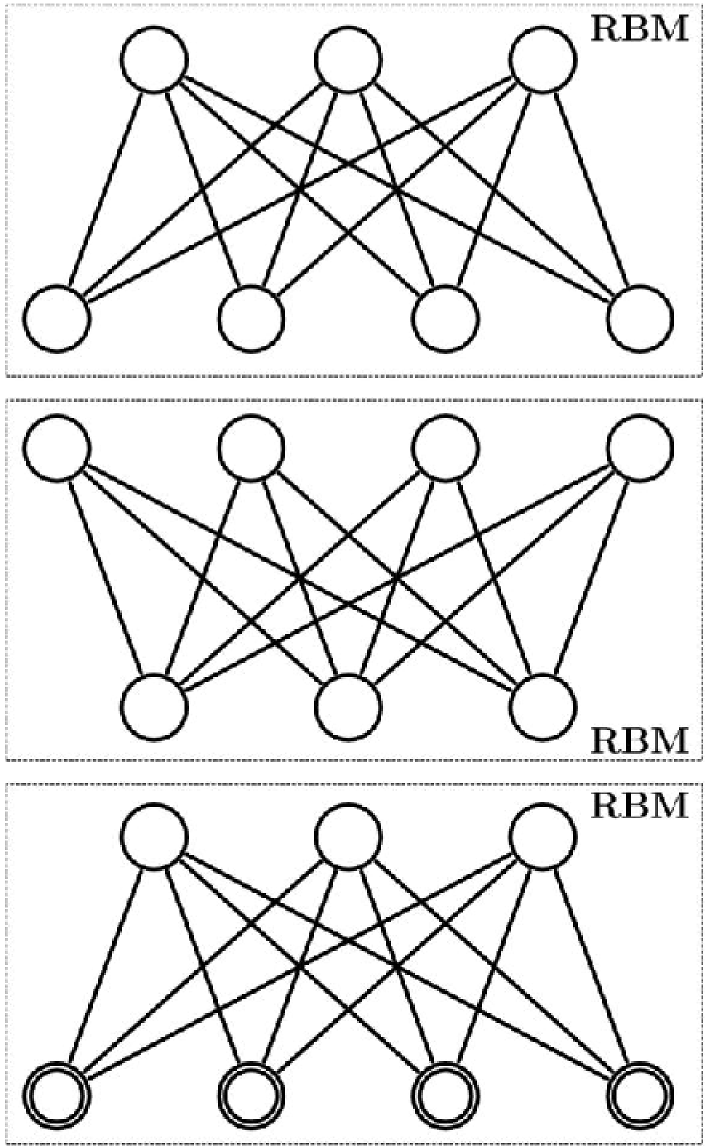
Deep Boltzmann Machine
[undirected]

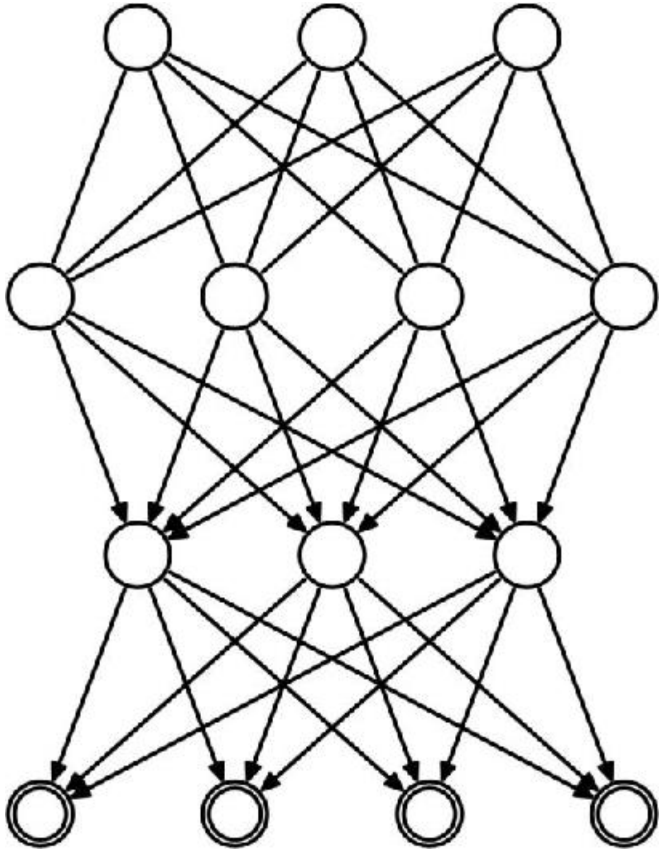


Deep Belief Network
[mixed]

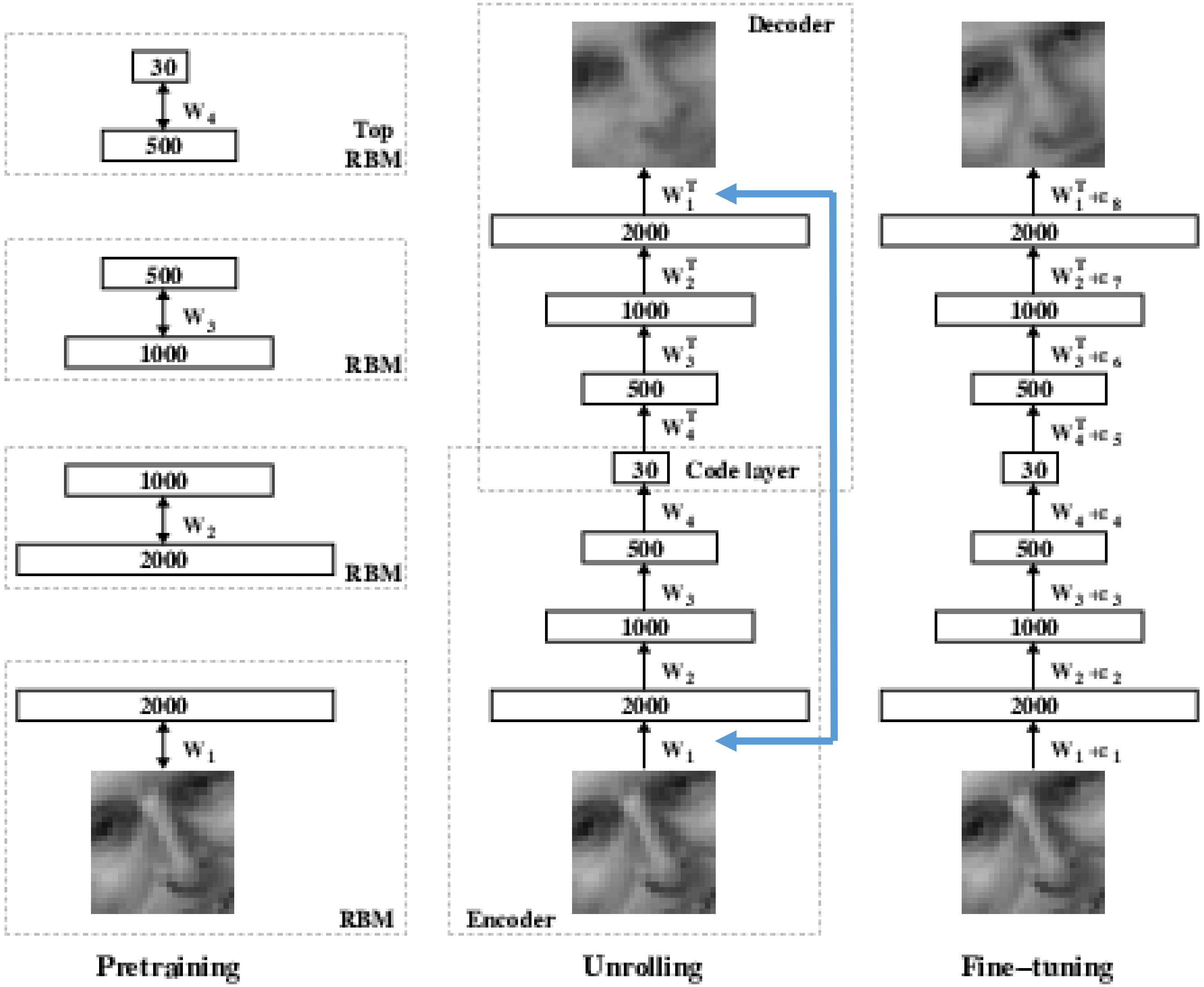
Deep Belief Networks



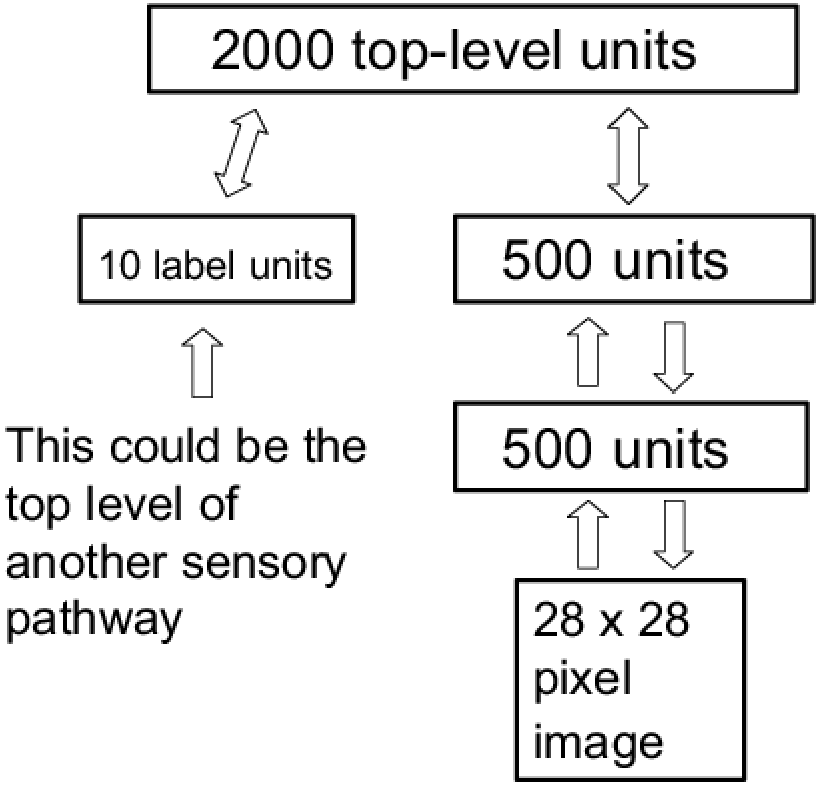




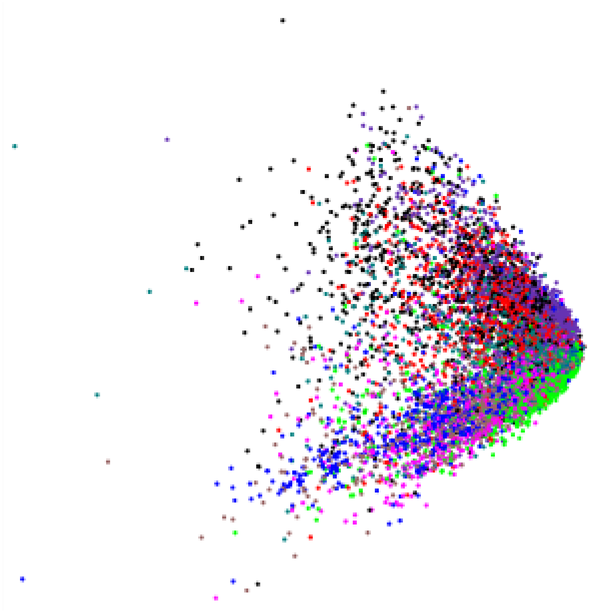
Training a Deep Auto-Encoder



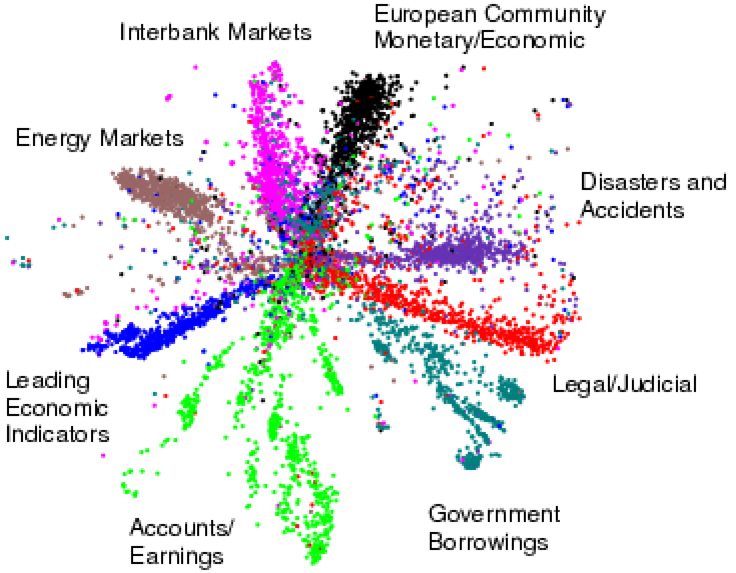
Deep Belief Network for MNIST



2D Visualization of Reuters Data

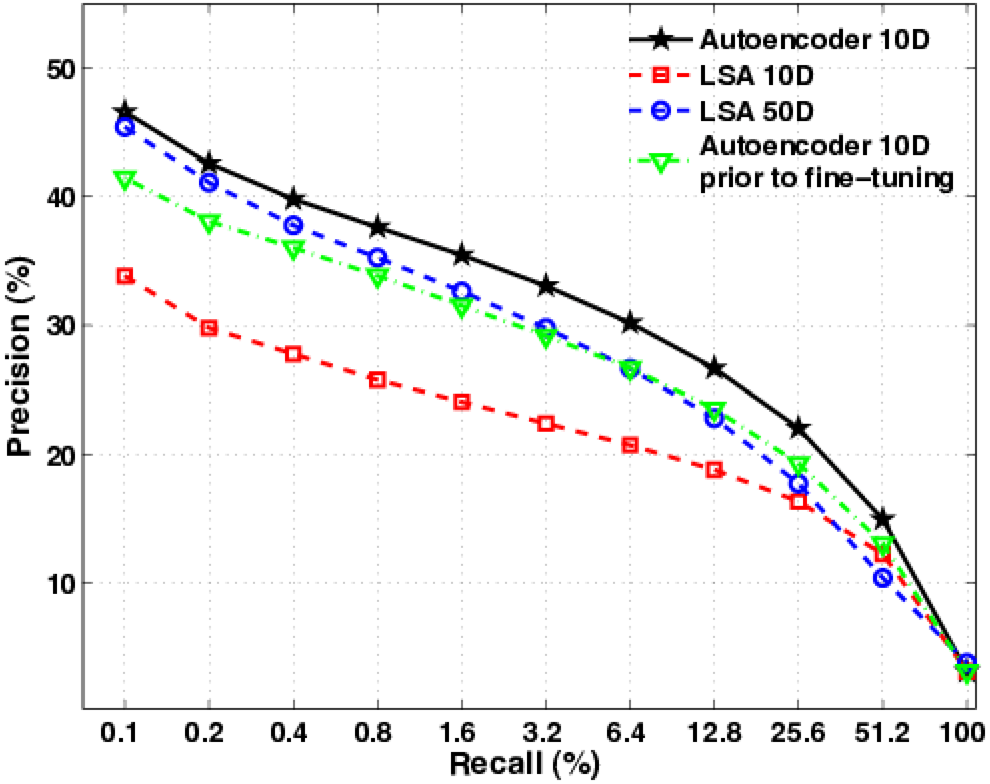


LSA

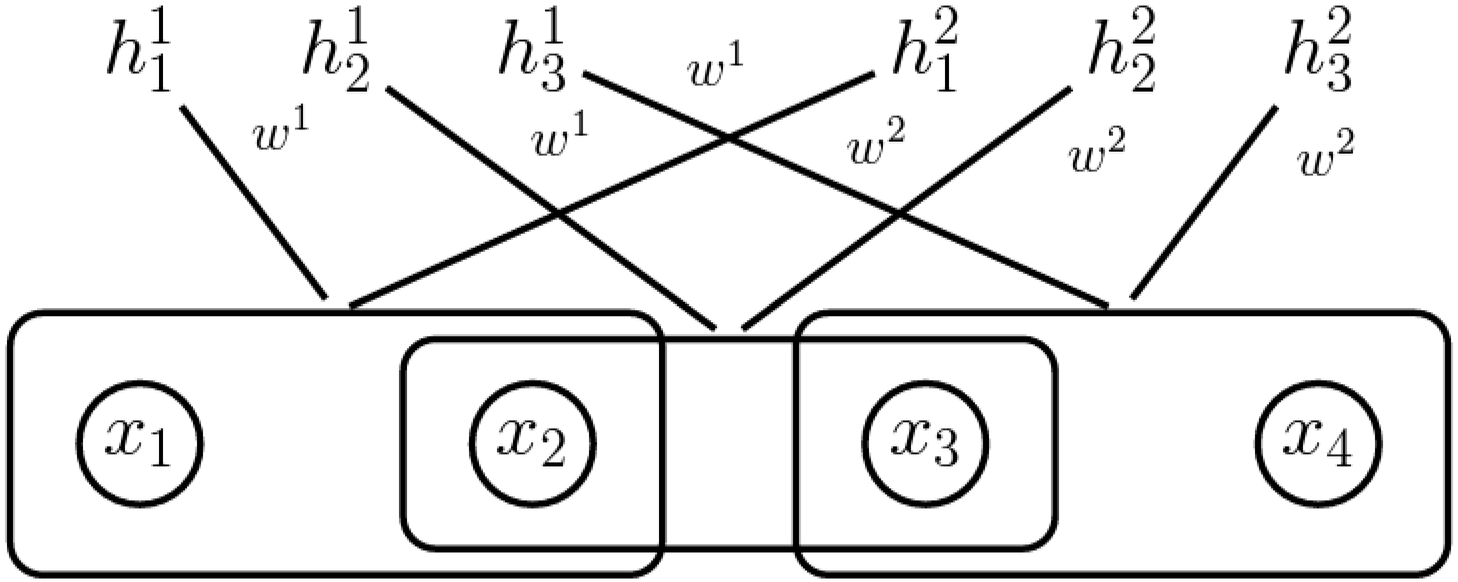


Auto-encoder

Precision-Recall Curves for Reuters Data

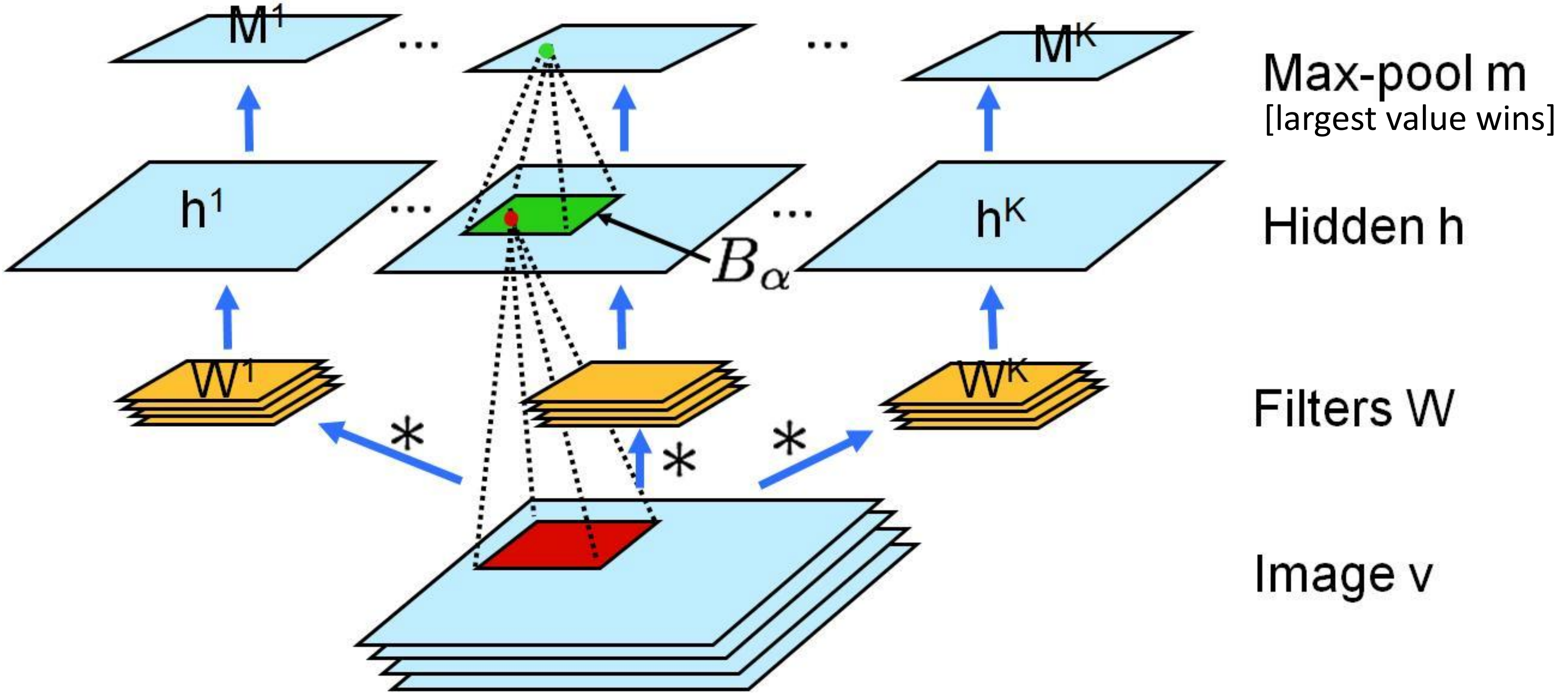


1D Convolutional RBM

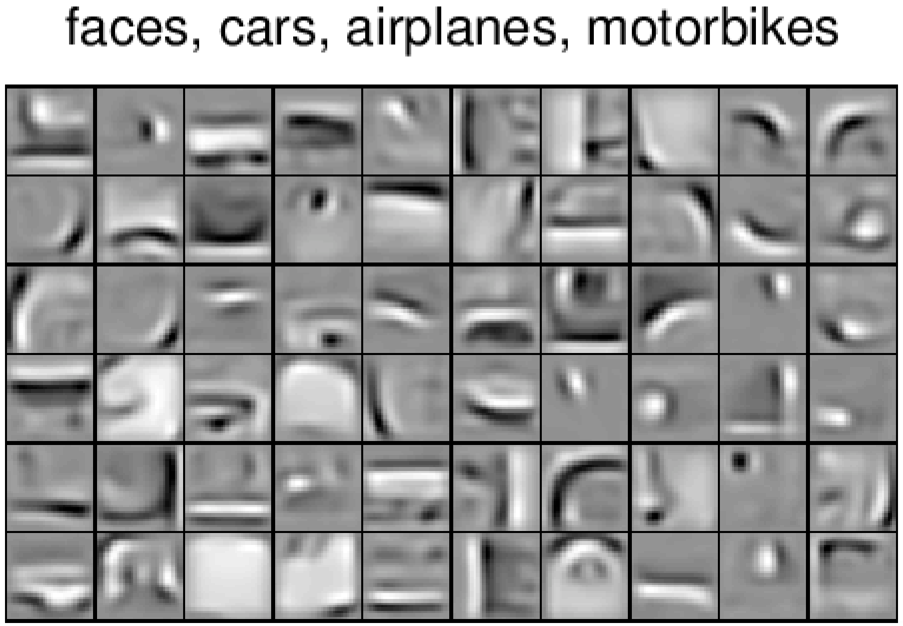


convolution: combining 2 input functions to create a 3rd function
[used to combine a “signal” (stream or image) with a “filter” (weight matrix)]

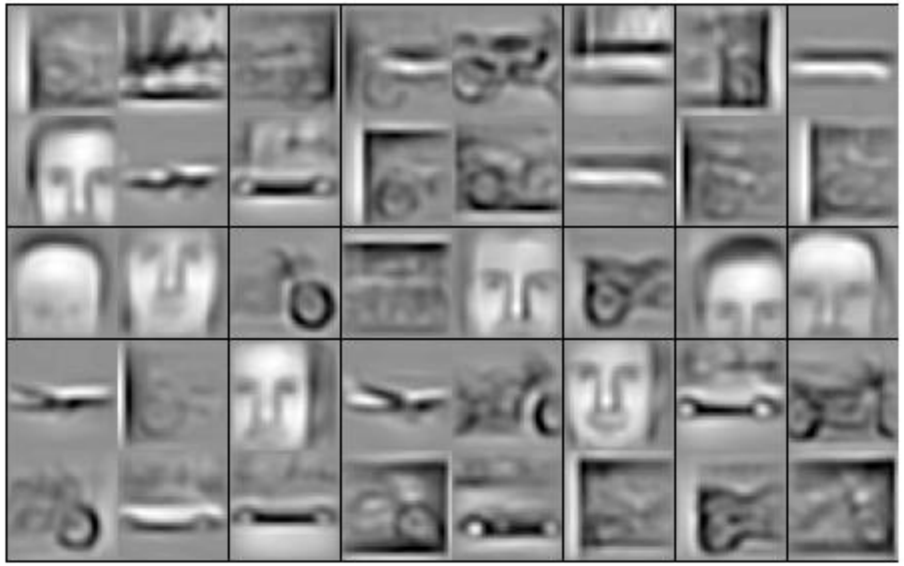
2D Convolutional RBM with Max Pooling



Visualization of Filters for Convolutional DBN



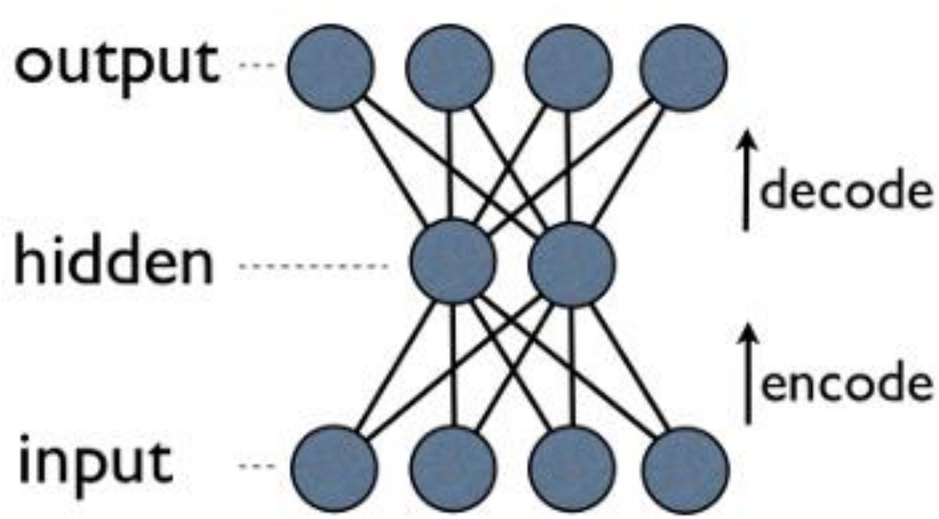
lower-level features



higher-level features

RBMs, Auto-Encoders, and Convolutional Neural Networks

Auto-Encoder

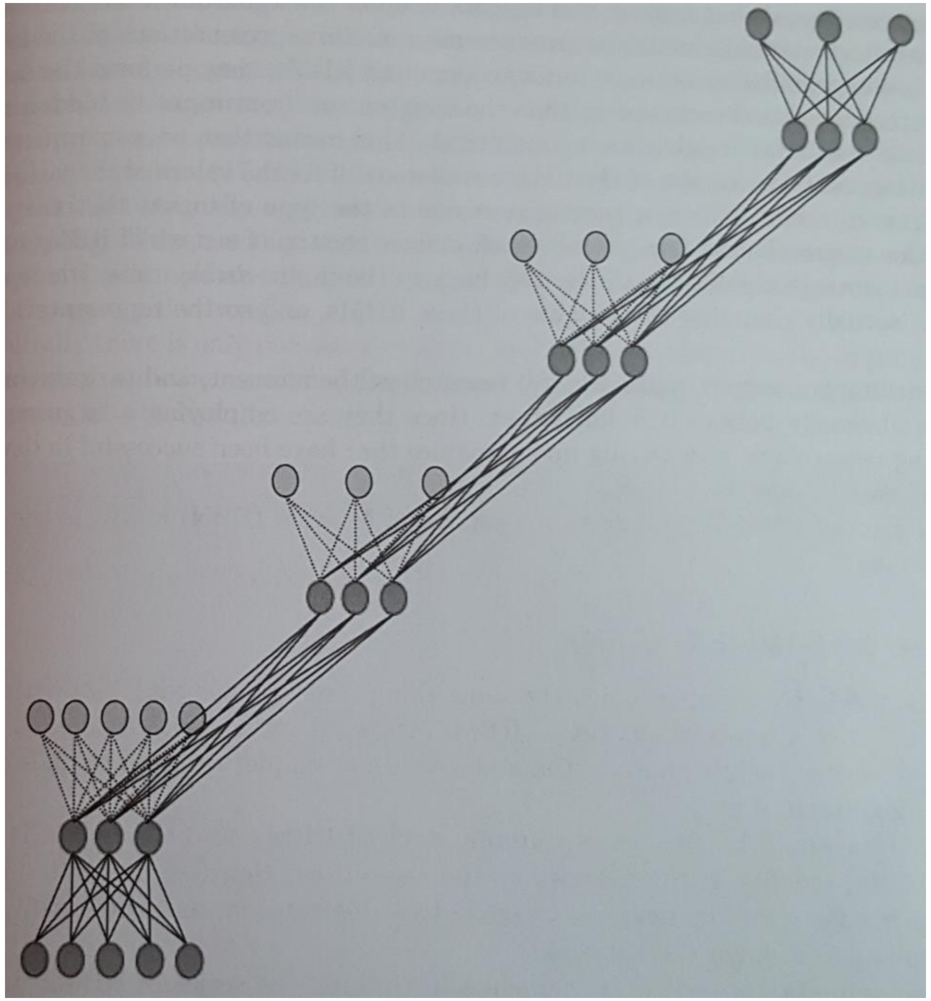


Encoded: abstract representation

Under-complete (bottleneck) versus over-complete representation

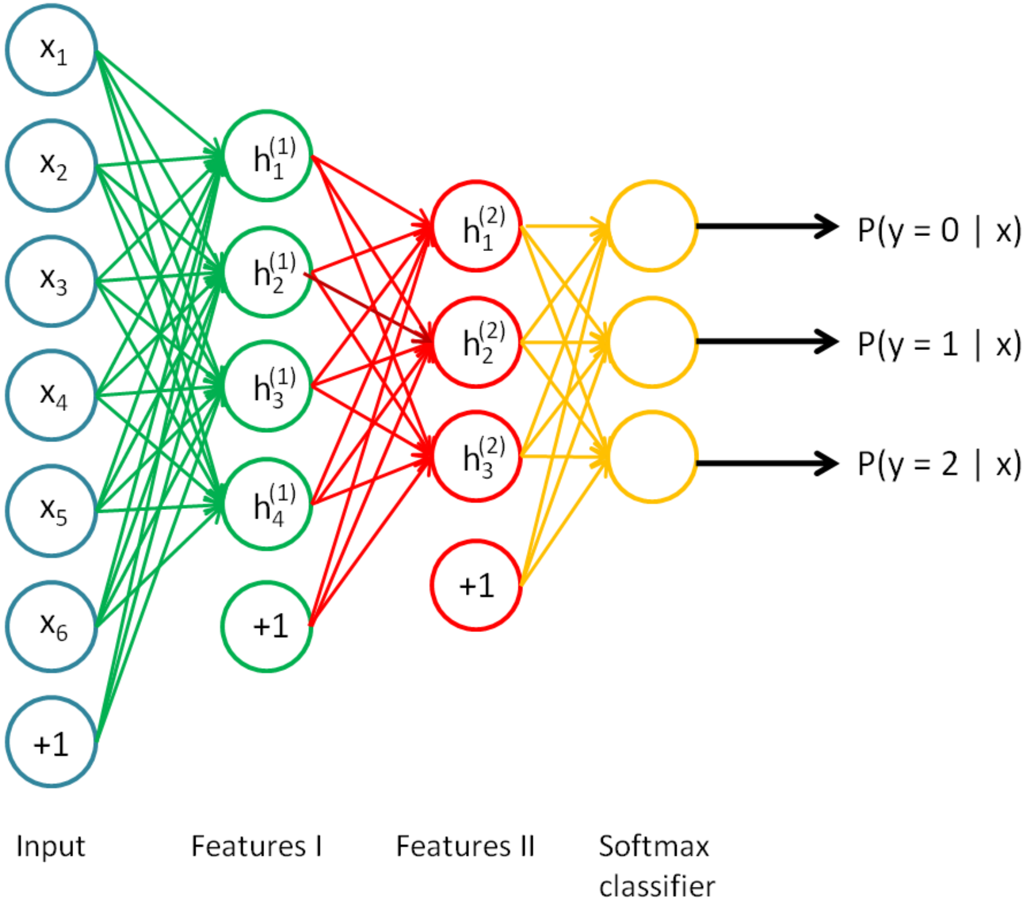
Can be trained using backpropagation

Stacked Auto-Encoders



https://seat.massey.ac.nz/personal/s.r.marsland/MLBook.html

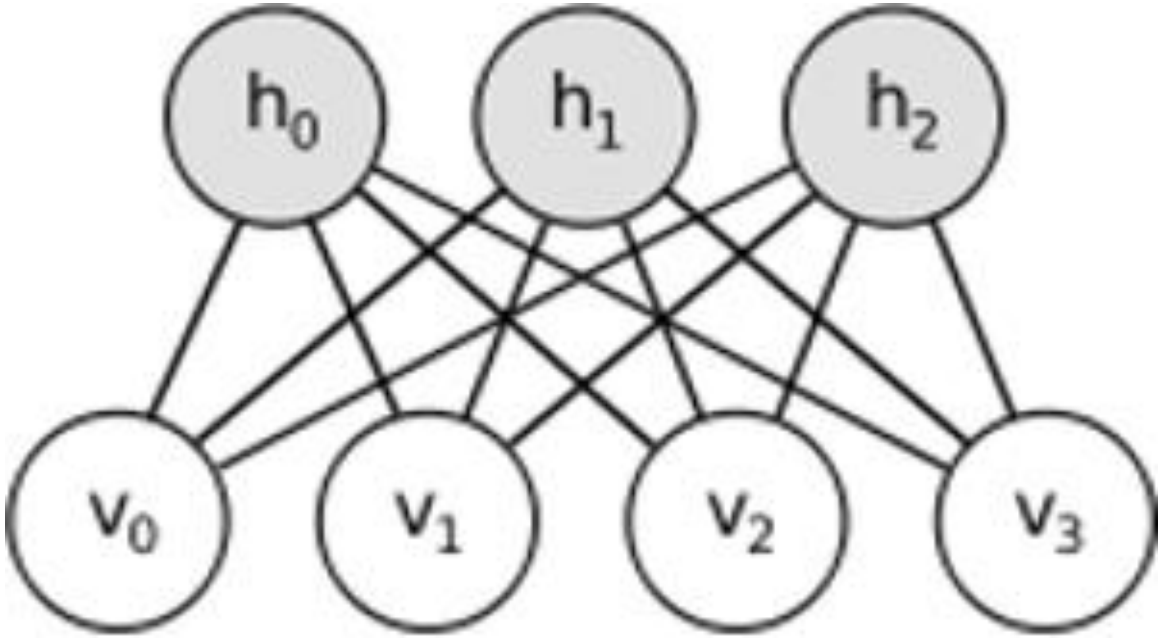
Stacked Auto-Encoder: Alternative Visual



"cleaner" image;
because it doesn't show
the "output" layers used
to construct the
hidden layers

http://ufldl.stanford.edu/wiki/images/5/5c/Stacked_Combined.png

Restricted Boltzmann Machine



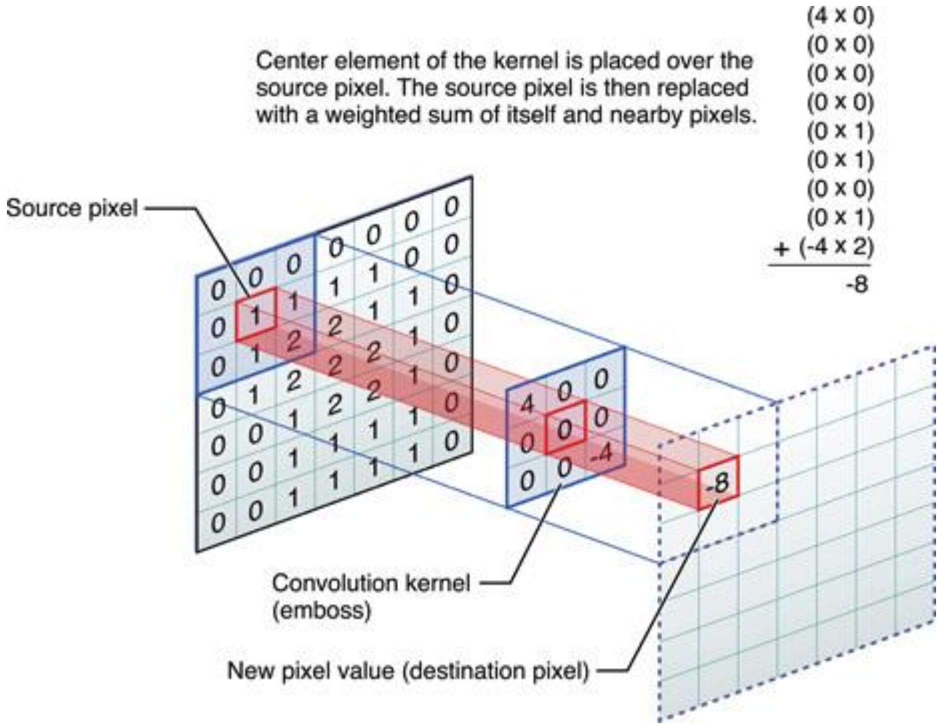
Hidden layer
[an abstract representation]

Visible layer

Generative model

Can be trained using contrastive divergence

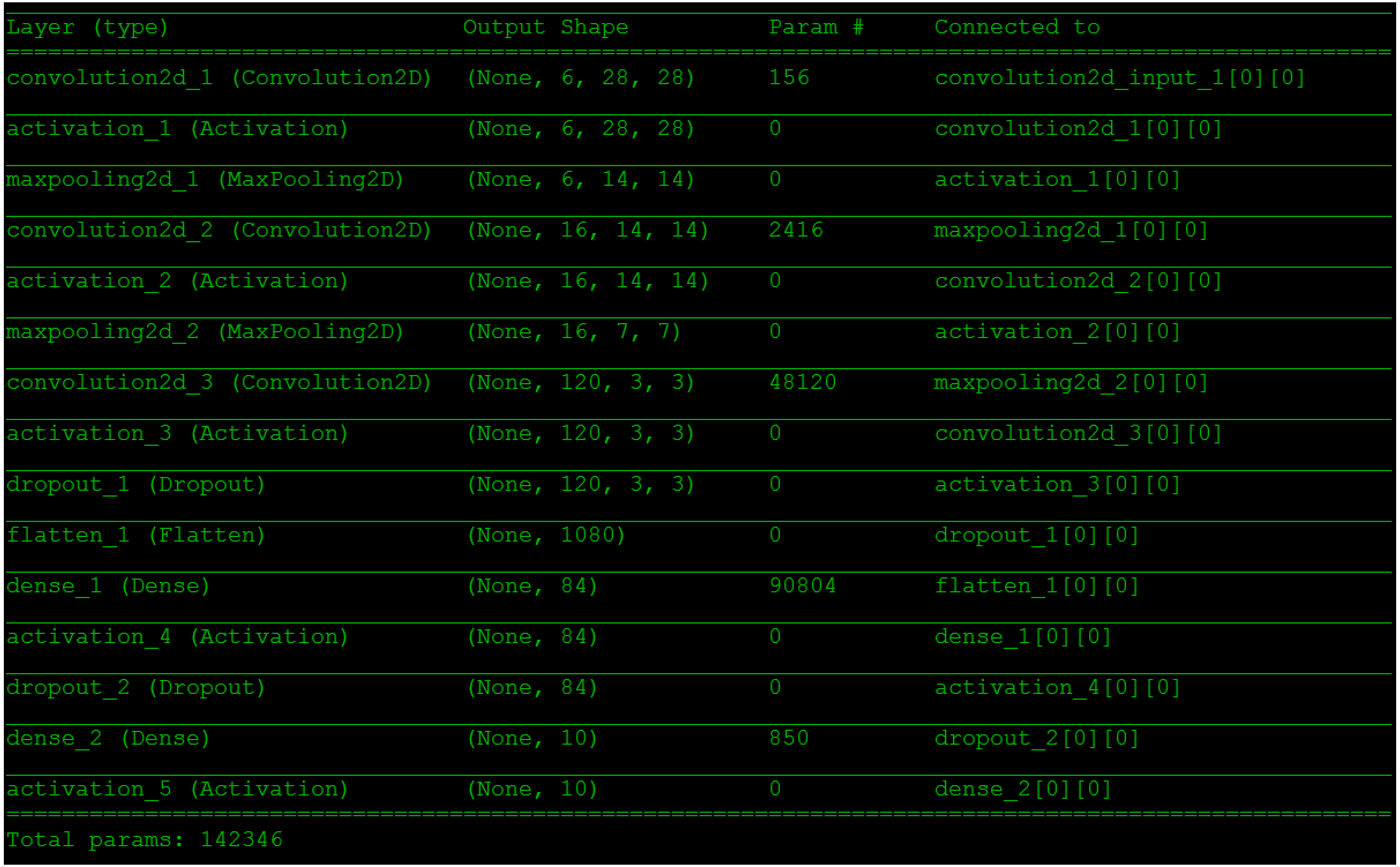
Convolution Example


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)

Filter response is maximized when image pixel set resembles filter

SUPER IMPORTANT:
Filters can be learned!

Convolutional Neural Network Example



```
____________________________________________________________________________________________________
Layer (type)                      Output Shape          Param #     Connected to
====================================================================================================
convolution2d_1 (Convolution2D)   (None, 6, 28, 28)     156         convolution2d_input_1[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)         (None, 6, 28, 28)     0           convolution2d_1[0][0]
____________________________________________________________________________________________________
maxpooling2d_1 (MaxPooling2D)     (None, 6, 14, 14)     0           activation_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)   (None, 16, 14, 14)    2416        maxpooling2d_1[0][0]
____________________________________________________________________________________________________
activation_2 (Activation)         (None, 16, 14, 14)    0           convolution2d_2[0][0]
____________________________________________________________________________________________________
maxpooling2d_2 (MaxPooling2D)     (None, 16, 7, 7)      0           activation_2[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)   (None, 120, 3, 3)     48120       maxpooling2d_2[0][0]
____________________________________________________________________________________________________
activation_3 (Activation)         (None, 120, 3, 3)     0           convolution2d_3[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)               (None, 120, 3, 3)     0           activation_3[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)               (None, 1080)          0           dropout_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                   (None, 84)            90804       flatten_1[0][0]
____________________________________________________________________________________________________
activation_4 (Activation)         (None, 84)            0           dense_1[0][0]
____________________________________________________________________________________________________
dropout_2 (Dropout)               (None, 84)            0           activation_4[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                   (None, 10)            850         dropout_2[0][0]
____________________________________________________________________________________________________
activation_5 (Activation)         (None, 10)            0           dense_2[0][0]
====================================================================================================
Total params: 142346
```

$6 * (5 * 5 + 1) = 156$

$28 / 2 = 14$

$16 * (6 * 5 * 5 + 1) = 2416$

$14 / 2 = 7$

$120 * (16 * 5 * 5 + 1) = 48120$

$7 - 5 + 1 = 3$

$120 * 3 * 3 = 1080$

$84 * (1080 + 1) = 90804$

$10 * (84 + 1) = 850$

Examples

- Unsupervised learning: rbm_auto-encoder.ipynb
- CNN: convolutional_neural_network.ipynb
- http://deeplearning.net/tutorial/ code

```
$ wget -r -np http://deeplearning.net/tutorial/code/
$ cd deeplearning.net/tutorial
$ mkdir data
$ cd data
$ wget http://www.iro.umontreal.ca/~lisa/deep/data/mnist/mnist.pkl.gz
$ cd ../code
$ time python logistic_sgd.py; time python mlp.py
$ time python SdA.py; time python DBN.py    # greedy, layer-wise training
```

Simple MNIST Performance Comparison

method	time (mm:ss)	error rate
logistic	00:10	7.49%
mlp	65:52	1.65%
SdA	179:13	1.42%
DBN	71:41	1.36%
CNN	~ 4:00	0.85%