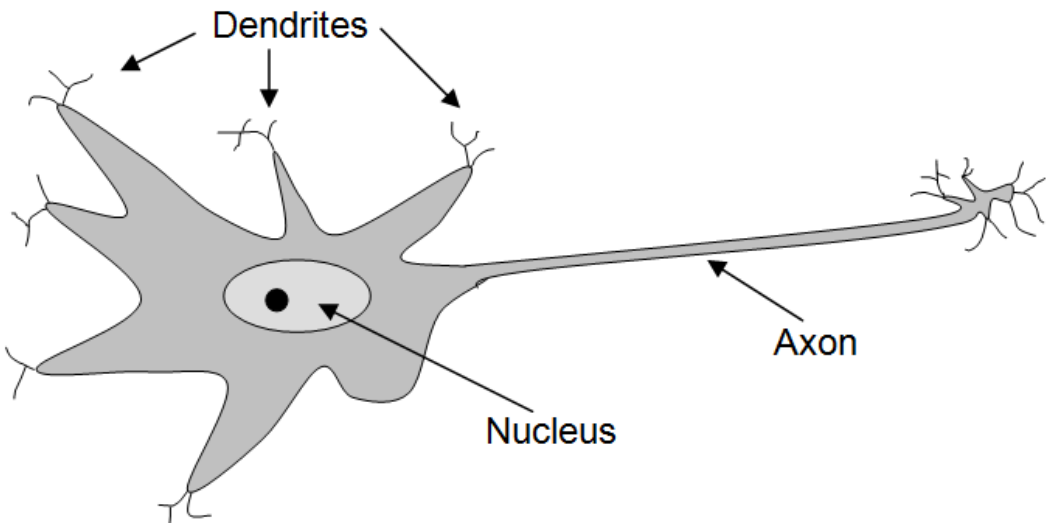
Introduction to deep learning

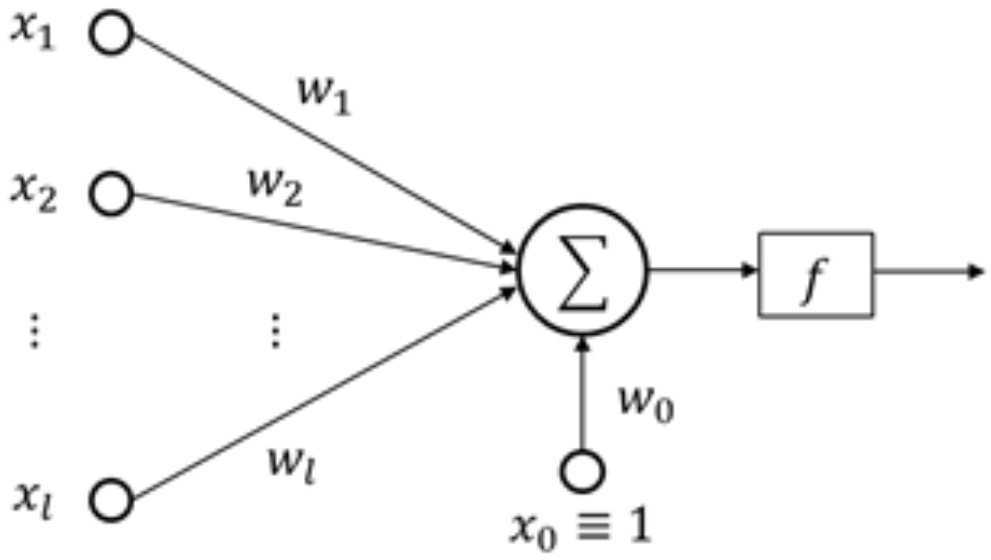
Introduction to artificial neural network (ANN or NN)

- A machine learning algorithm for classification, clustering, function approximation, etc.
- Motivated by how biological neural network learn and process information.
  - Cerebral cortex contains 10^{11} neurons that are deeply connected into a massive network.
  - Each neuron is connected to $10^3 - 10^4$ other neurons.
  - A neuron can receive information from other neurons, process the information, and then pass it to other neurons.



Simple perceptron model

- A perceptron model is the simplest, single-nueron model for supervised learning.
- Training data contains: inputs x_i, $i = 1, \ldots, l$. Each x_i is a n-vector; and output d, a n-vector. The output can be continuous (regression) or binary (classification).
- An activation function f is specified by user for binary outcome.



Learn the perceptron model

The learning algorithm is based on gradient search to minimize the squared loss: $\sum_{j=1}^{n}(d_j - y_j)^2$, where $y_j = f(w_0 + \sum_{i=1}^{l} w_i x_{ij})$.

1. Initialize w's to random numbers. Then at iteration r:
2. Compute $y_j^r = f(w_0^r + \sum_{i=1}^{l} w_i^r x_{ij})$.
3. Update weights by:
   - $w_0^{(r+1)} = w_0^r + \gamma_1^r \sum_{j=1}^{n}(d_j - y_j^r) f'(z_j^r)$.
   - $w_i^{(r+1)} = w_i^r + \gamma_2^r \sum_{j=1}^{n}(d_j - y_j^r) f'(z_j^r) x_{ij}$,

   Here $z_j^r = w_0^r + \sum_{i=1}^{l} w_i^r x_{ij}$. γ_1^r and γ_2^r are the learning rate (step size).

Artificial neural network

ANN is a glorified perceptron model with multiple neurons and (optionally) multiple layers (has at least one hidden layer of neurons).

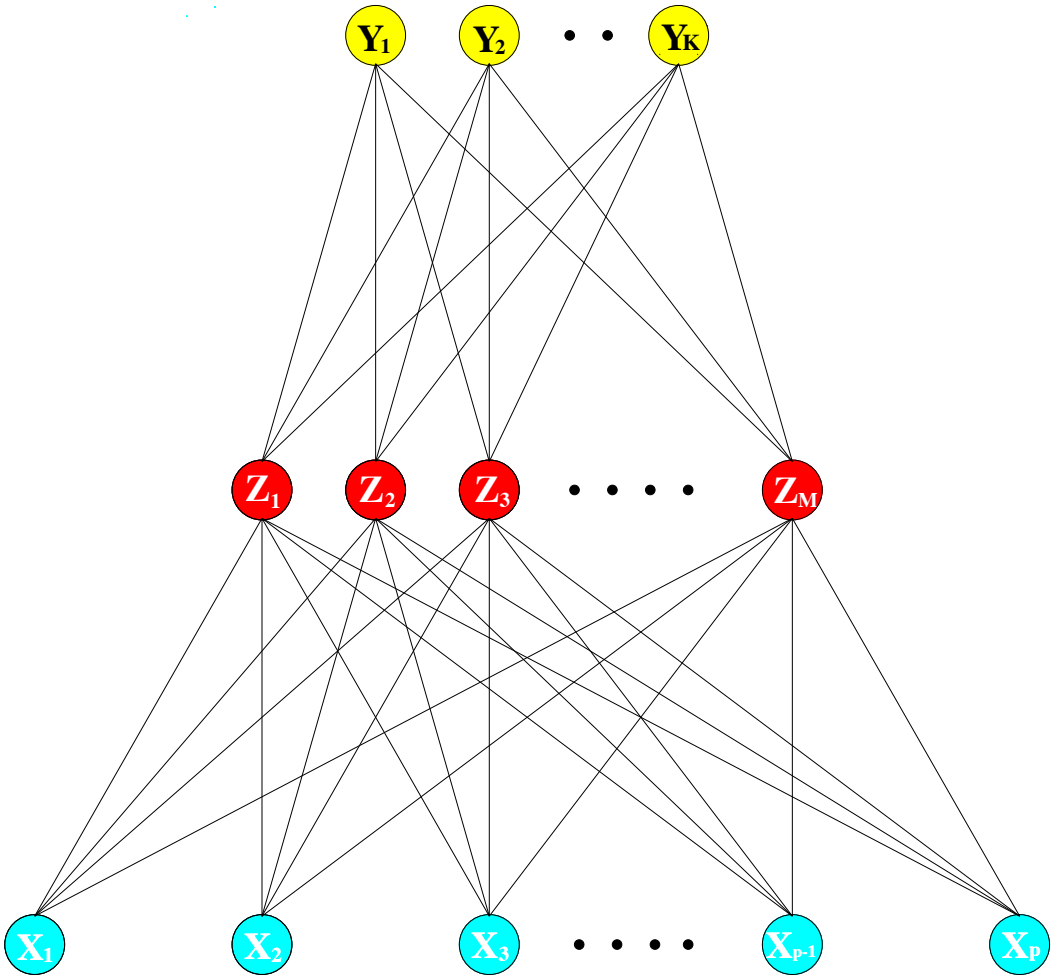


Figure is from ”elements of statistical learning” chapter 11, figure 11.2

- The neural network has input (X), output (Y), and hidden variable Z.
- It works for continuous or categorial outcomes. For continuous, there will be one Y at the top. For K-class classification, there will be K nodes at the top, each coded as 0/1.
- In an ANN, a mathematical neuron works the same as a perceptron. It receives a number of inputs, computes the weighted sum and then generate outputs through activation functions.
- The mathematical model for an ANN is

$$
\begin{aligned}
z_m &= \sigma(\alpha_{0m} + \alpha_m^T X),\ m = 1, \ldots, M. \\
T_k &= \beta_{0k} + \beta_k^T Z,\ k = 1, \ldots, K. \\
f_k(X) &= g_k(\mathbf{T}),\ k = 1, \ldots, K.
\end{aligned}
$$

Here, σ is activation function, such as sigmoid.

In K-class classification, usually use softmax function

$$
g_k(\mathbf{T}) = \frac{e^{T_k}}{\sum_l e^{T_l}}
$$

ANN model fitting

Parameters in an ANN are:

- α_{0m}, α_m: $M(P+1)$
- β_{0m}, β_k: $K(M+1)$

Objective function of an ANN is

- For continuous outcome, residual sum of squares:

$$R(\theta) = \sum_{k=1}^{K} \sum_{i=1}^{N} (y_{ik} - f_k(x_i))^2$$

- For categorical outcome, cross-entropy : (or NLL, negative log likelihood)

$$R(\theta) = -\sum_{k=1}^{K} \sum_{i=1}^{N} y_{ik} \log f_k(x_i)$$

- Model fitting is done by “back propagation algorithm”.

Back propagation algorithm

Essentially a gradient descent algorithm.

- Initialization: pick random weights.
- Forward: given weights, compute predicted values.
- Backward: update the weights, using first derivatives as direction. The first derivatives can be obtained using chain rule.

Detailed derivations are skipped. Please refer to "*Elements of Statistical Learning*".

Note:

- Be careful of overfitting: the model is too flexible and has too many parameters.
- Regularization technique (e.g., add L1 penalty) can be used to stabilize the model fitting.
- Number of neurons and the number of layers are set by user.
- Compared with the perceptron model, ANN produces non-linear boundary. Each hidden node is a linear classifier. With adequate numbers of neurons and hidden layers, arbitrary decision boundary can be formed.

Neural network in R

The `neuralnet` package in R provides functions to train a neural network.

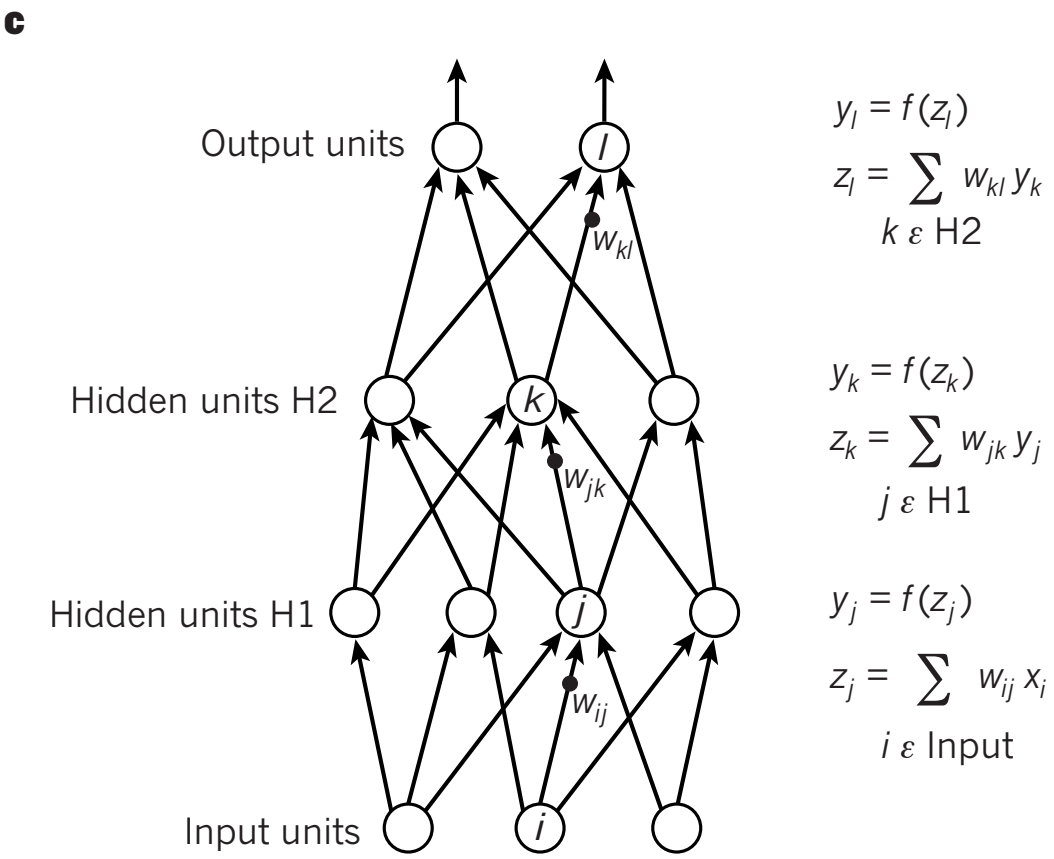
```
library(neuralnet)
data(infert, package="datasets")

## fit NN
fit <- neuralnet(case~parity+induced+spontaneous, infert)
predicted <- fit$net.result[[1]]>0.5
table(infert$case, predicted)
   predicted
    FALSE TRUE
  0   143   22
  1    37   46

## compare with SVM
library(e1071)
fit.svm <- svm(case~parity+induced+spontaneous, infert)
predicted <- predict(fit.svm)>0.5
table(infert$case, predicted)
   predicted
    FALSE TRUE
  0   149   16
  1    43   40
```

Deep learning

- Appeared quite a while ago (1980's), but gained tremendous attention only recently, mostly due to the increased computational power and collection of training data.
- Becomes a social buzz word after the Google AlphaGo beats the world champion 4-1 in a set of five GO games.

Deep Architecture

There are different types, but the most common is the feed-forward multilayer neural network.



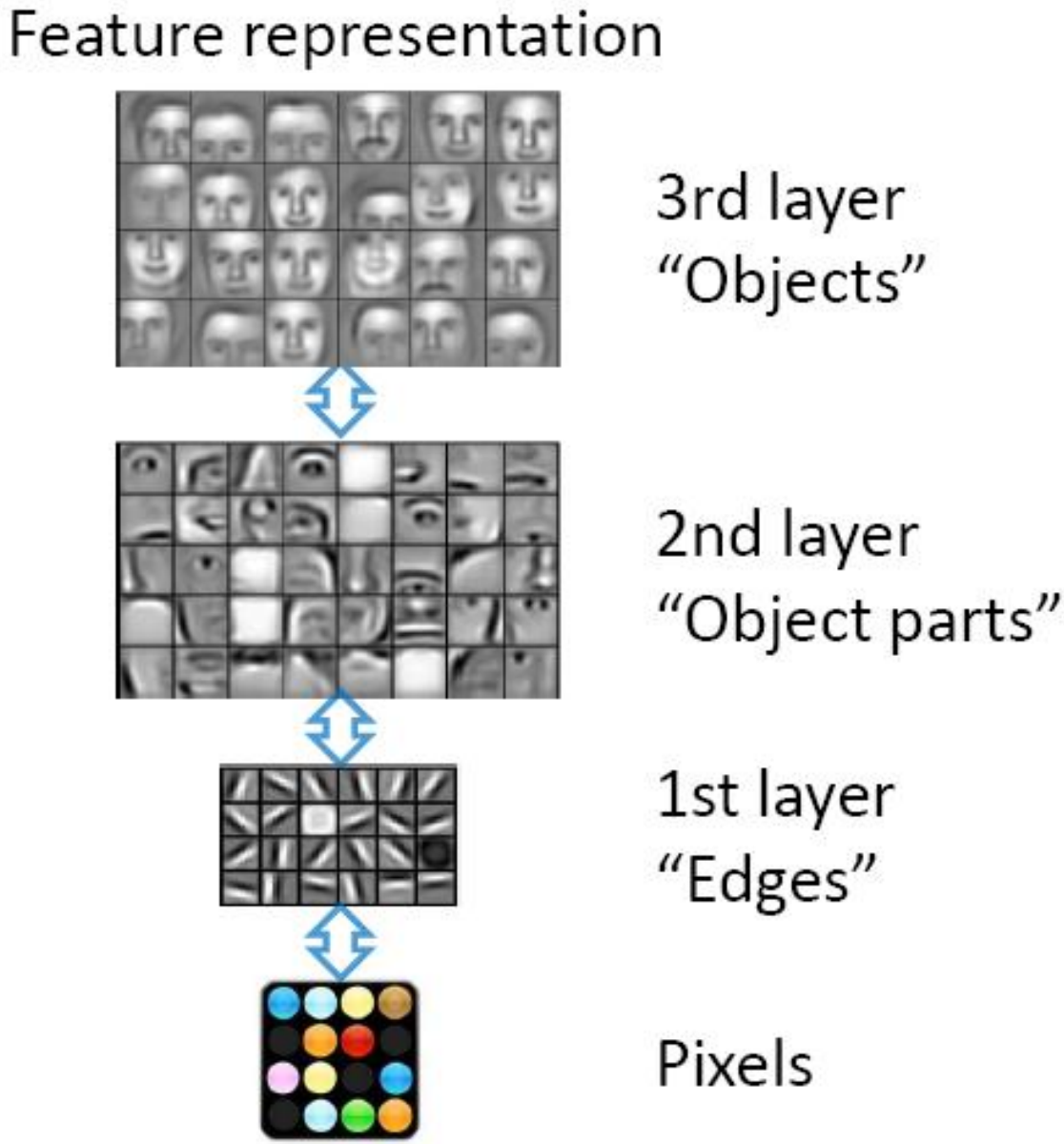
Why go deep, since one can approximate any function as close as possible with shallow architecture?

- Deep machines are more efficient for representing certain classes of functions.
- So deep architecture trades breadth for depth (more layers, but less neurons in each layer).
- Demo: Tensorflow Playground: http://playground.tensorflow.org/

Advantages:

- Traditional machine learning algorithm requires feature extraction from data: Data $\rightarrow$ Feature $\rightarrow$ Model. Finding the correct features is critical in the success of a ML model.
- In complex pattern recognition/prediction problem (such as audio/image recognition, natural language processing), the input signals are highly non-linear and feature extraction is difficult.
- Deep learning can better capture the non-linearity in the data, thus potentially automate the feature selection step.

Example: DL in image recognition



Example: DL in image recognition (cont.)

In image recognition problem, the inputs are color intensity values for all pixels in a picture. Tradition method that directly link pixels to outcome doesn't work well, because the higher order interactions (patterns) are not captured efficiently.

In deep learning (such as convolutional neural network):

- Layer 1: presence/absence of edge at particular location and orientation.
- Layer 2: motifs formed by particular arrangements of edges; allows small variations in edge locations
- Layer 3: assemble motifs into larger combinations of familiar objects
- Layer 4 and beyond: higher order combinations

Key: the layers are not designed, but learned from data using a general-purpose learner.

Train a deep neural network

Back propagation does not work well if randomly initialized.

It was shown that deep networks trained with back propagation (without unsupervised pretraining) perform worse than shallow networks.

	train.	valid.	test
DBN, unsupervised pre-training	0%	1.2%	1.2%
Deep net, auto-associator pre-training	0%	1.4%	1.4%
Deep net, supervised pre-training	0%	1.7%	2.0%
Deep net, no pre-training	.004%	2.1%	2.4%
Shallow net, no pre-training	.004%	1.8%	1.9%

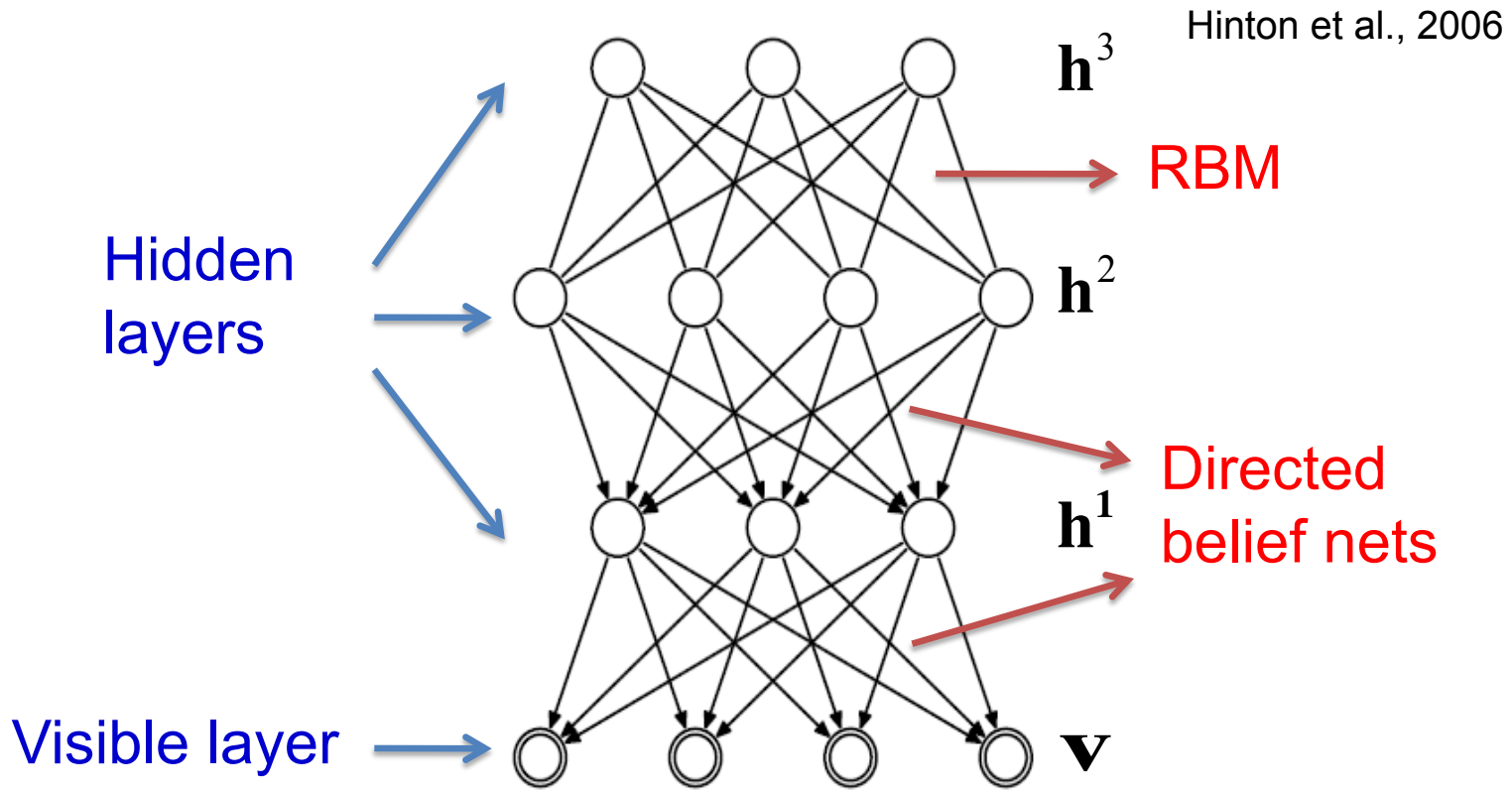
(Bengio et al., NIPS 2007)

Problems with Back Propagation: gradient is progressively getting more dilute below top few layers.

Deep Network Training

Hinton *et al.*, (2006) **A fast learning algorithm for deep belief nets** proposes greedy layer-wise training for training a deep belief network (DBN).

DBN is a type of deep neural network, which can be viewed as a stack of simple, unsupervised networks such as restricted Boltzmann machines (RBMs).



DBN Greedy Training

Let v be the input data (visible layer).

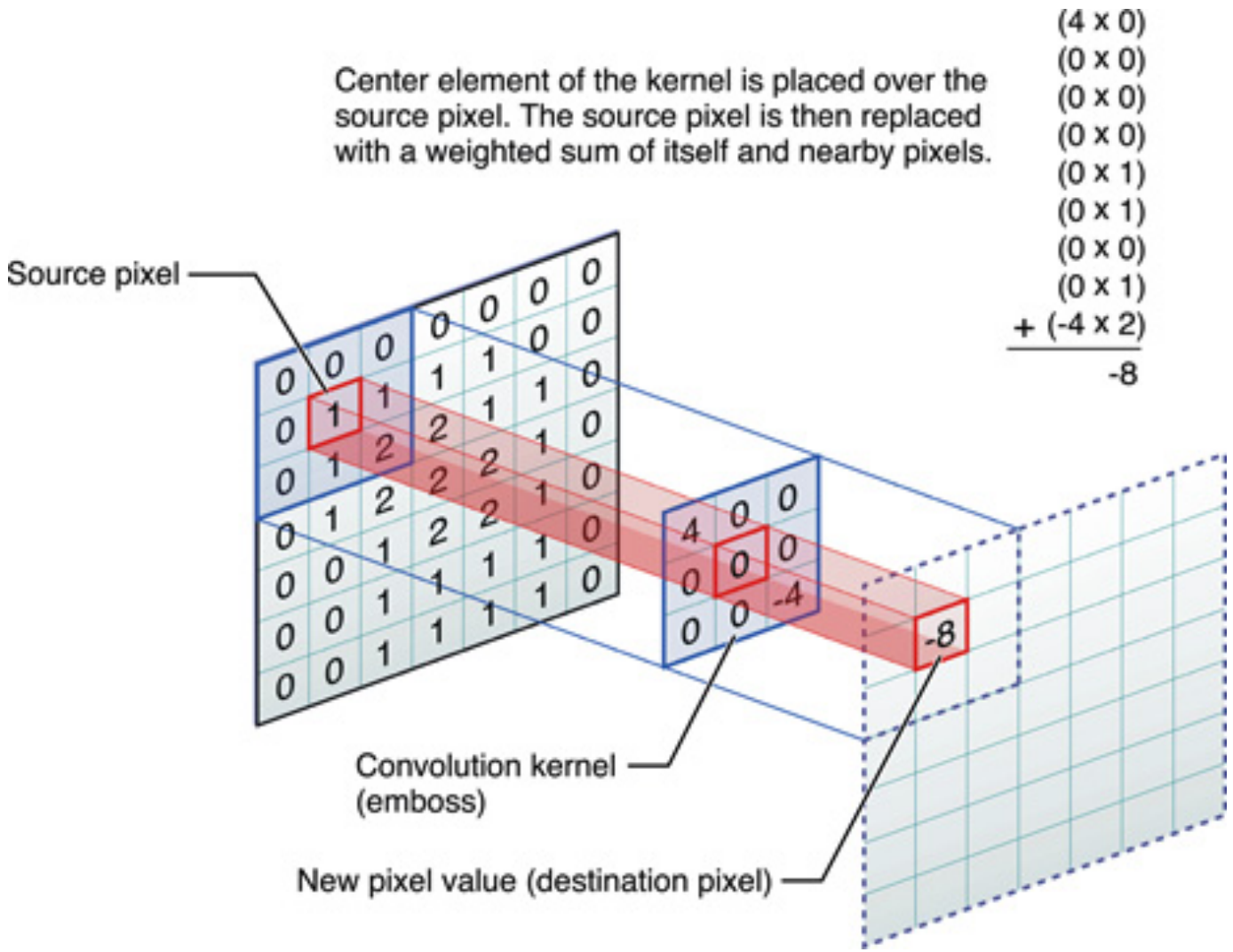
- First construct an RBM with input layer v and hidden layer h^1. The trained RBM provides $p(h^1|v)$.
- Obtain a set of realization of hidden layer h^1 (denoted by $\tilde{h}^1$) based on the trained RBM. One can either sample from $p(h^1|v)$, or compute $E[p(h^1|v)]$.
- With $\tilde{h}^1$ and hidden layer h^2, train another RBM, and so on.
- Once all parameters are estimated, perform supervised top-down training (backprop) to refine the parameters.

Convolutional neural network (CNNs / ConvNets)

- CNN is a type of feed forward neural network.
- Widely applied to data where nearby values are correlated, for example, images (pixel values are spatially correlated), sound (frequencies are temporally correlated), etc.
- The network are stacked by convolutional and pooling layers for feature extraction.
- The final layer of the network is a fully connected layer to connect the extracted features to the output.

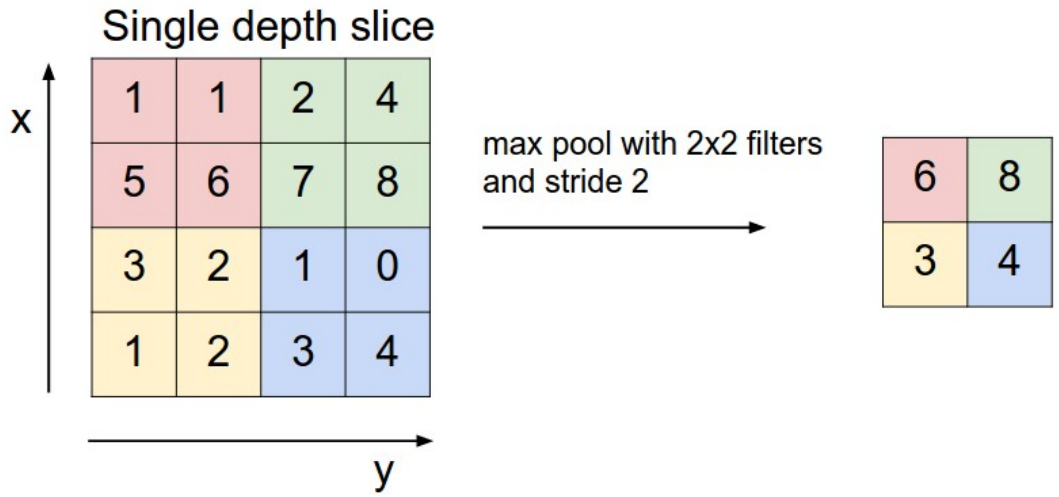
Convolutional layer

- Each convolutional layer contains a set of filters (kernels). For image, the convolutional layer is a 2D matrix with weights.
- Number of filters in each convolutional layer is determined by user.
- The convolution step is to scan the input data and compute the local weighted sum. For image, the step is like a 2D moving sum.
- The weights in the filters are learnable, and will be estimated through model training.
- The estimated weights capture the “motifs” in the input data. Filters in lower layers capture more basic features (like the edges). Higher layer filters capture more complicated features (like the object parts).
- The convolution is based on an important assumption that the same features can appear in different location of the input data (“**shared weights**”). This greatly reduce the number of parameters compared to a fully connected network.

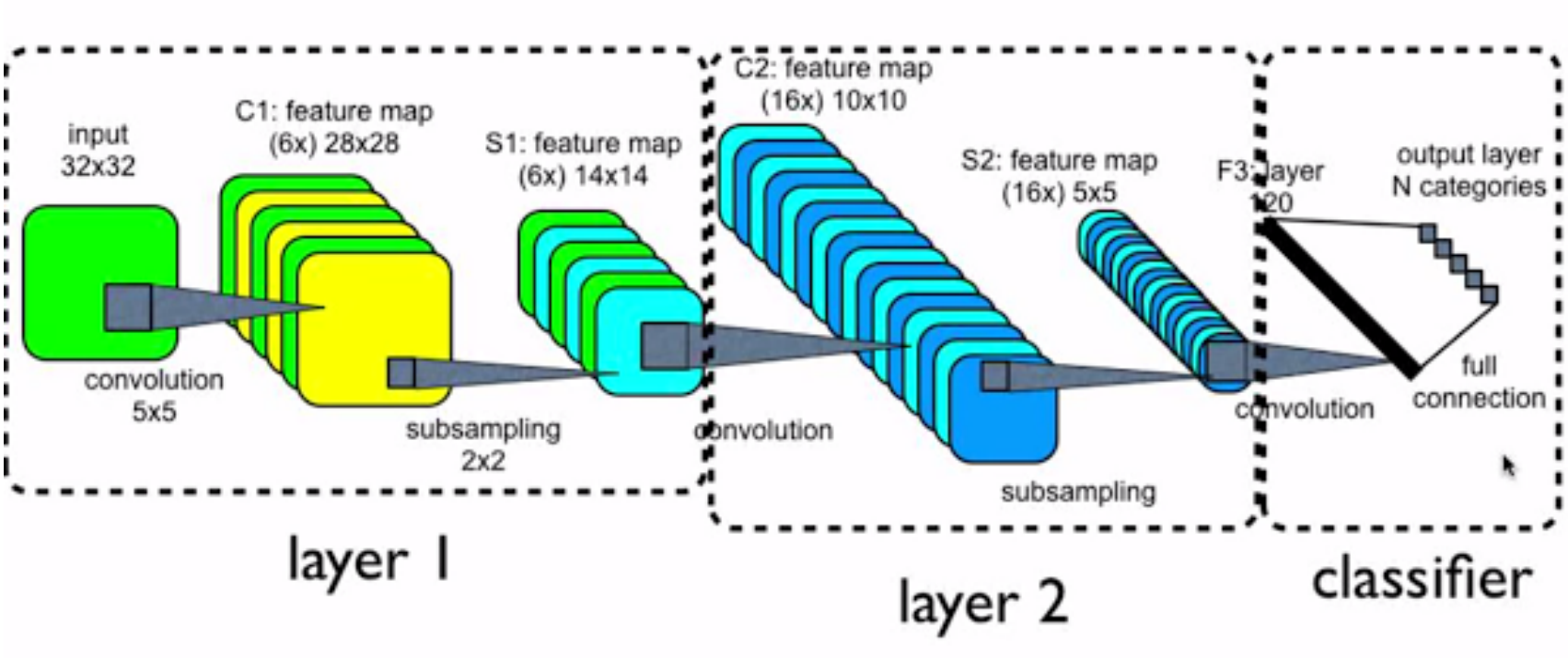

Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel (emboss)
New pixel value (destination pixel)

Pooling layers

- After the convolution, scan the data and apply a pooling function (usually max).
- Used to reduce the spatial size of the representation, reduce the amount of parameters and computation, and alleviate overfitting.
- For example, a pooling layer with filters of size 2x2 downsamples the input by 2 along both width and height, discarding 75% of the data.
- In backpropagation, one only routes the gradient to the input that had the highest value in the forward pass. So it's important to keep track of the index of the max activation during the forward pass of a pooling layer.



ConvNets diagram



- The model learning is based on back propagation.
- Pre-training is helpful.
- Large number of parameters (often millions). Require large training set.

Deep learning software package

Most of the popular deep learning software packages are written in Python, for example,

- Tensorflow by Google: https://www.tensorflow.org.
- Theano: http://deeplearning.net/tutorial/.

The R development for deep learning lags behind, but gradually catches up: There are a few packages:

- MXNetR seems to be a really good one: http://dmlc.ml/rstats/2015/11/03/training-deep-net-with-R.html.
- Other available ones on CRAN include `RNN`, `LSTM`, `darch`, `deepnet`.

Summary

- Deep learning is a powerful technique in the machine learning field.
- So far the major areas of application include speech recognition, image classification, natural language processing.
- Demonstrate superior performance when there are many highly nonlinear patterns in the data.
- Application in biostatistics/bioinformatics field is relatively fewer so far, perhaps because
  - Training data size (subjects) is still too small.
  - Biological knowledge, in the form of existing networks, are already explicitly used, instead of being learned from data. They are hard to beat with a limited amount of data.
  - Statisticians are not familiar with DL.