Pattern Recognition and Machine Learning

James L. Crowley

ENSIMAG 3 - MMIS Fall Semester 2016/2017
Lessons 9 11 Jan 2017

Artificial Neural networks

Outline

Using notation and figures from the Stanford Deep learning tutorial at: http://ufldl.stanford.edu/tutorial/

Notation

x_d	A feature. An observed or measured value.
$\vec{X}$	A vector of D features.
D	The number of dimensions for the vector $\vec{X}$
$\{\vec{X}_m\}$ $\{y_m\}$	Training samples for learning.
M	The number of training samples.
N	Spatial extent (size) of the convolutional unit (NxN)
$a_{i,j}^{k}$	A "feature map" of *k* features at each image position *P(i,j)*
$D^{(l)}$	is the number of activation units in layer *l*.
$\hat{\rho}_j = \frac{1}{M}\sum_{m=1}^{M} a_{j,m}^{(2)}$	Sparsity: The average activation of hidden unit j for the training set

$$\sum_{j=1}^{D^{(2)}} KL(\rho \,\|\, \hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j}$$ Kullback-Leibler Divergence

Convolutional Neural Networks.

Convolutional Neural Networks take inspiration from the Receptive Field model of biological vision systems proposed by Hubel and Weisel in 1968 to explain the organization of the visual cortex. By probing the visual cortex of cats with electrodes, Hubel and Weisel discovered that the visual cortex was composed of retinal maps of visual features. Each retinal map is an image of the pattern projected on the retina expressed with correlation of a visual feature at a particular size (scale) and orientation. The collection retinal feature maps serves as an intermediate representation for recognition.

Fully connected Networks.

A fully connected network is a network where each unit at level $l+1$ receives activations from all units at level l.

If there are $D^{(l)}$ units at level l and $D^{(l+1)}$ units are level $l+1$ then a fully connected network requires learning $D^{(l)}\cdot D^{(l+1)}$ parameters. While this may be tractable for small examples, it quickly becomes excessive for practical problems, as found in computer vision or speech recognition.

For example, a typical image may have 1024 x 1024 = 2^{20} pixels. If we assume, say a 512 x 512 =2^{18} hidden units we have 2^{38} parameters to learn for a single class of image pattern. Clearly this is not practical (and, in any case unnecessary)

A common solution is to perform learning using a limited size window, and to use all possible windows as training data. We have already seen this principle with the use of a sliding window approach with the Viola Jones detector.

This leads to a technique where we fix a window size at NxN input units and use all possible, overlapping, windows of size NxN from our training data to train the network.

We then use the same learned weights with every hidden cell. The resulting operation is equivalent to a "convolution" of the learned weights with the input signal.

This approach is reasonable for image and speech signals because both images and speech signals have two interesting theoretical properties: They are <u>local</u> and <u>stationary</u>.

1) Local. Local means that (most of) the required information can be found within a limited sized neighborhood of the signal. In fact, image information tends to be multi-scale, but this can be easily accommodated using multi-scale signal techniques using a scale invariant pyramid. Such a representation is "local" at multiple scales, with low-resolution scales providing context for higher resolution. This can be referred to as "multi-local".

2) Stationary. A stationary signal is a random (unknown) signal whose joint probability density function does not change when shifted in time (speech) or space (image). Image and Speech signals tend to have stationary statistics. Thus the same processing can be applied to every possible (overlapping) window.

There are exceptions to both rules, but these can be handled with established techniques.

What Window Size?

What Window Size for a feature in a Convolutional Neural Network? This tends to depend on the problem. It is not uncommon to see tutorials proposal 5 x 5 image windows. This may be fine for illustration, but likely to be far from optimal. The impressive results in category learning were obtained with a 2D image window of 11 x 11.

A minimum reasonable size is N=7. For smaller windows, the Fourier transform of the window function (the digital Sync function) dominates the spectrum and hides information. N=9 is a better choice as the window effects at are negligible.

However, there is the problem of scale invariance. In computer vision, patterns can occur at many scales. The solution is to use a scale invariant pyramid, providing invariant representation at sampled set of scaled. The window size should accommodate all scale changes between pyramid steps. This is generally between N=11 and N=15. The actual choice depends on available training data, computing time for learning and the scale steps used in the pyramid.

For today, consider that N=11 is a reasonable size.

Convolutional Neural Network for an image.

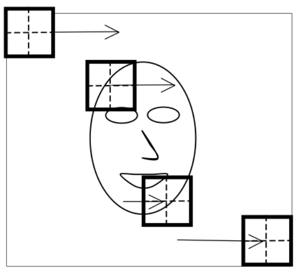
Convolutional Neural Network (CNN) can be used as feature detectors for image analysis. When used with images, a CNN provides K features at each pixelusing convolution with K hidden layers (or kernels). Each feature will be computed as a weighted sum of the pixels within an N x N window R(i,j).

Let us assume an image of J rows and I columns, where each pixel has 3 color values. Each pixel *(i,j)* is a color vector, $\vec{P}(i,j)$, represented by 3 integers between 0 and 255 representing Red, Green and Blue.

$$\vec{P}(i,j) = \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

In the literature on CNNs, the colors are referred to as "channels", and the number of channels is called the Depth (D)l. We will refer to the position of receptive field as its center. As a result we prefer odd values for N.

The CNN will compute K filters (or kernels) for each window $\underline{R_{ij}(u,v)}$ of size NxNx D that fits within the image. If we consider the position of the window as its upper left corner, then for each position from $i=N/2, j=N/2$ *to* $i = I-N/2, j=J-N/2$:



$$R_{ij}(u,v) = P(i+u-1, j+v-1) \quad \text{for} \quad i=N/2, j=N/2 \text{ to } i = I-N/2, j=J-N/2$$

The features can be learned as hidden layers using back-propagation, using a training set where each window is labeled with a target class. For example, with the FDDB training set, each possible NxN window $R_{ij}(u,v)$ would be an training vector $\vec{X}_m$ of size $D^{(1)}=D\cdot N^2$. The target value y_m would be 1 if the center pixel of the window is in the face ellipse and 0 Otherwise.

The result is a "feature map" of k features at each position *(i,j)*

$$a_{i,j,k}^{(2)} = f(\sum_{u,v} w_k^{(1)}(u,v)R_{i,j}(u,v) + b_k^{(1)})$$

Note that written as a convolution, the formula would be

$$a_k(i,j) = f(\sum_{u,v} w_k(u,v)R(i-u,j-v) + b_k)$$

Hyperperameters:

CNNs are typically configured with a number of "hyper-parameters":

Depth: This is the number D of channels for each image pixel.

Stride: Stride is the step size, S, between window positions. By default it may be 1, but for larger windows, it is possible define larger step sizes.

Spatial Extent: This is the size, NxN of the receptive field.

Zero-Padding: Size of region at the border of the feature map that is filled with zeros in order to preserve the image size (typically N/2).

Pooling

Pooling is a form of non-linear down-sampling that partitions the image into non-overlapping regions and computes a representative value for each region.

Pooling is typically performed over contiguous regions of the image. In this case, the stride equals the pooling window size. The CNN feature image is partitioned into small non-overlapping rectangular regions, typically of size 2x2 or 4x4.

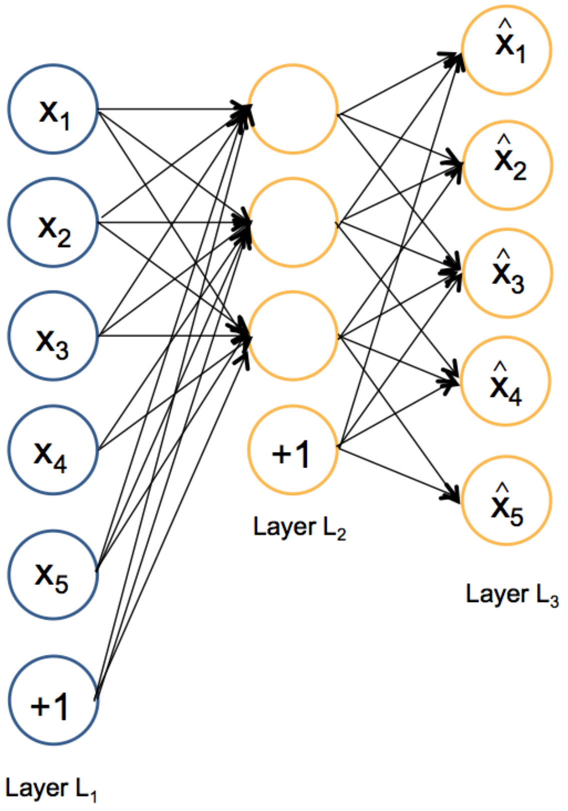
Several non-linear functions can be used. These include Max, Average, Median, and Histograms. Max pooling seems to be the most popular.

For example, the SIFT operation, in Computer vision, uses local histograms over a 4x4 window.

AutoEncoders

An autoencoder is an unsupervised learning algorithm that uses back-propagation to learning a sparse set of features for describing the training data. Rather than try to learn a target variable, y_m, the auto-encoder tries to learn to reconstruct the input X using a minimum set of features.

The effect is similar to the use of Principal components analysis on neighborhoods, (as we saw in lecture 4). However, the auto-encodeur provides a more appropriate basis set for recognition, while principle components anaysis is appropriate for reconstruction.



Using the notation from our 2 layer network, given an input feature vector $\vec{X}_m$ $\{w_{ji}^{(1)}, b_j^{(1)}\}$ and $\{w_{kj}^{(2)}, b_k^{(2)}\}$ such that $\vec{a}_m^{(3)} = \hat{X}_m \approx \vec{X}_m$ using as few weights as possible. Note that $D^{(3)} = D^{(1)}$.

When the number of hidden units $D^{(2)}$ is less than the number of input units, $D^{(1)}$,

$\vec{a}_m^{(3)} = \hat{X}_m \approx \vec{X}_m$ is necessarily an approximation.

The error for back-propagation for each unit is $\delta_{k,m}^{(3)} = a_{k,m}^{(3)} - a_{i,m}^{(1)} = a_{k,m}^{(3)} - x_{i,m}$
For each component $x_{i,m}$ of the training sample $\vec{X}_m$

The Sparsity Parameter

The auto-encoder will learn weights subject to a sparseness constraints specified by a sparsity parameter $\hat{\rho}_j = \rho$, typically set close to zero. The sparsity parameter ρ is the average activation for the hidden units.

Using the notation from last week's lecture, the auto-encoder is described by:

Level 1: $\vec{X}_m = \begin{pmatrix} x_{1,m} \\ \vdots \\ x_{D^{(1)},m} \end{pmatrix}$

level 2: $a_{j,m}^{(2)} = f(\sum_{i=1}^{D^{(1)}} w_{ji}^{(1)} x_{i,m} + b_j^{(1)})$

level 3: $a_{k,m}^{(3)} = f(\sum_{j=1}^{D^{(2)}} w_{kj}^{(2)} a_{j,m}^{(2)} + b_k^{(2)})$

Desired output $\vec{a}_m^{(3)} = \begin{pmatrix} a_1^{(3)} \\ \vdots \\ a_D^{(3)} \end{pmatrix} = \hat{X}_m \approx \vec{X}_m$, with error $\delta_{k,m}^{(3)} = a_{k,m}^{(3)} - a_{i,m}^{(1)} = a_{k,m}^{(3)} - x_{i,m}$

The average activation $\hat{\rho}_j$ is computed as the average activation for each of the *D*$^{(2)}$ hidden units, *j=1 to D*$^{(2)}$ for the M training samples:

$$\hat{\rho}_j = \frac{1}{M} \sum_{m=1}^{M} a_{j,m}^{(2)}$$

The auto-encoder can be learned by back-propagation using a minor change to the cost function.

$$L_{sparse}(W, B; \vec{X}_m, y_m) = \frac{1}{2}(\vec{a}_m^{(3)} - \vec{X}_m)^2 + \beta \sum_{j=1}^{D^{(2)}} KL(\rho \parallel \hat{\rho}_j)$$

where $\sum_{j=1}^{s_2} KL(\rho \parallel \hat{\rho}_j)$ is the Kullback-Leibler Divergence.

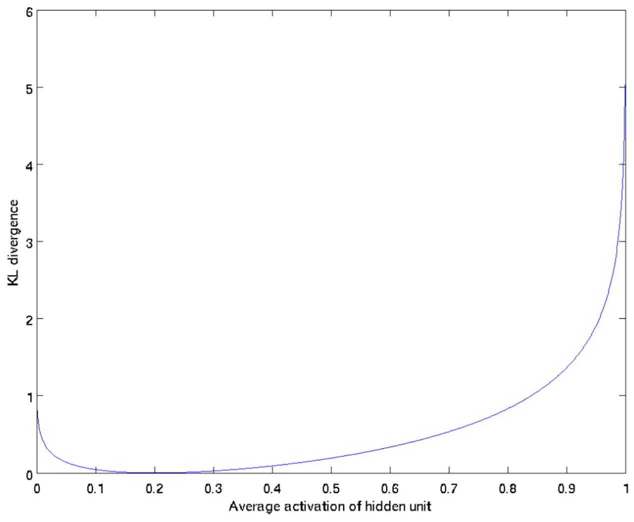
and β controls the weight of the sparsity parameter.

(Don't panic - this is easy to do).

Kullback-Leibler Divergence

The KL divergence between the desired and average activation is:

$$\sum_{j=1}^{D^{(2)}} KL(\rho \| \hat{\rho}_j) = \sum_{j=1}^{D^{(2)}} \left(\rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j} \right)$$



To incorporate the KL divergence into back propagation, we replace

$$\delta_j^{(2)} = \frac{\partial f(z_j^{(2)})}{\partial z_j^{(2)}} \sum_{k=1}^{D} w_{kj}^{(2)} \delta_k^{(3)}$$

with

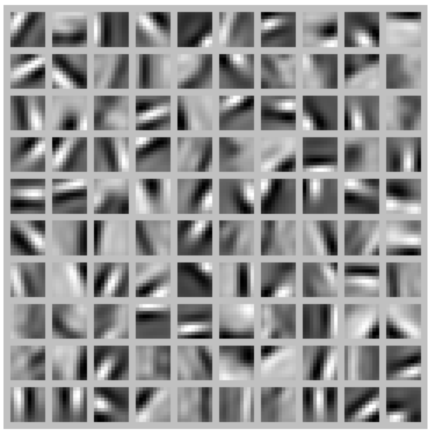
$$\delta_j^{(2)} = \frac{\partial f(z_j^{(2)})}{\partial z_j^{(2)}} \left(\sum_{k=1}^{D^{(1)}} w_{kj}^{(2)} \delta_k^{(3)} + \beta \left(-\frac{\rho}{\hat{\rho}_j} + \frac{1-\rho}{1-\hat{\rho}_j} \right) \right)$$

Note that you need the average activation $\hat{\rho}_j$ to compute the correction. Thus you need to compute a forward pass on all the training data, before computing the back-propagation on any of the training samples. This can be a problem if the number of training samples is large.

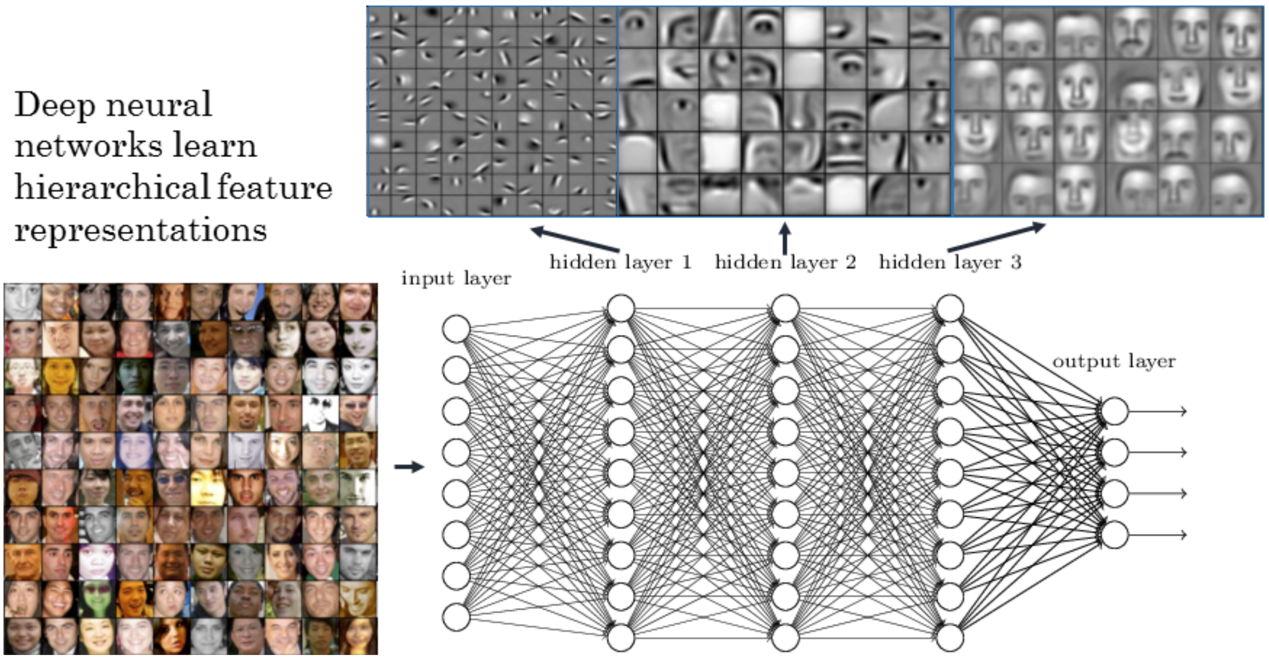
This forces the hidden units to become approximately orthogonal!
Thus the hidden units act as a form of basis space for the input vectors.

Examples of the Hidden Units given by Autoencoders

An example of 100 hidden units learned by a sparse auto-encoder from images:

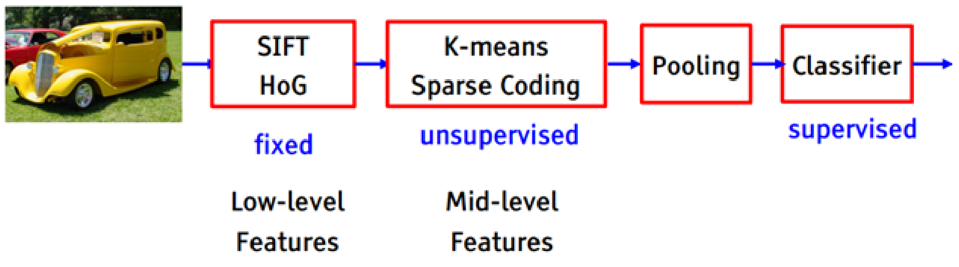


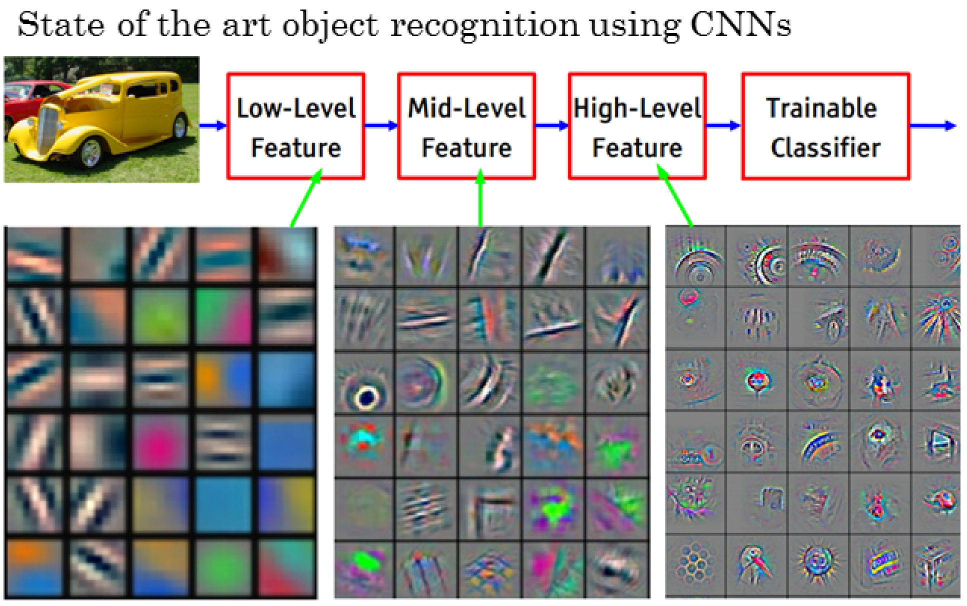
When applied at multiple levels and trained on face images this can give recognizable features:



or when trained on YouTube videos: Cats

when trained with car images:


SIFT
HoG
fixed
K-means
Sparse Coding
unsupervised
Pooling
Classifier
supervised
Low-level
Features
Mid-level
Features


State of the art object recognition using CNNs
Low-Level
Feature
Mid-Level
Feature
High-Level
Feature
Trainable
Classifier