Approximation theory

Gabriela Malenová

June 15, 2018

1 Ultimate goal

Approximate any function $f \in C_c^\infty(\mathbb{R}^n)$, $f : \mathbb{R}^n \mapsto \mathbb{C}^n$, by functions

$$v_j(\mathbf{x}) = e^{i(r_j + (\mathbf{x}-\mathbf{q}_j)\cdot\mathbf{p}_j + \frac{1}{2}(\mathbf{x}-\mathbf{q}_j)\cdot M_j(\mathbf{x}-\mathbf{q}_j))},$$

where $r_j \in \mathbb{R}, \mathbf{p}_j, \mathbf{q}_j \in \mathbb{R}^n$ and M_j a matrix such that $\Im M_j$ is positive definite. More precisely, we are to find $a_j, r_j, \mathbf{p}_j, \mathbf{q}_j, M_j, N$ for $j = 1, \dots N$ such that

$$g(\mathbf{x}) = \sum_{j=1}^{N} a_j v_j(\mathbf{x}),$$

and $\|f - g\|_2$ is minimized.

2 Simplification

Let us first consider f real and $r_j, \mathbf{p}_j, \Re M_j \equiv 0$ so that v_j is the Gaussian function

$$v_j(\mathbf{x}) = z_j^{-1} e^{-\frac{1}{2}(\mathbf{x}-\mathbf{q}_j)\cdot M_j(\mathbf{x}-\mathbf{q}_j)},$$

where z_j is chosen such that v_j is normalized. The number of terms N is fixed. This model is called the *Gaussian mixture model*. For f non-negative and normalized, algorithms based on unsupervised learning, in particular *expectation-maximization* offer a powerful tool.

2.1 Expectation-maximization

Let $X = \{\mathbf{x}_i\}_{i=1}^M$ denote a list of points in $\mathbb{R}^n$ and denote $f_i = f(\mathbf{x}_i)$. Without loss of generality we assume that $\sum_{i=1}^M f_i = 1$. Here, z_j are the normalization constants such that $\sum_{i=1}^M v_j(x_i) = 1$ for all $j = 1, \dots N$ and hence also $\sum_{j=1}^N a_j = 1$. Hence, g and v_j are probability measures on X.
The probablity that point $\mathbf{x}_i$ was generated from Gaussian v_j, denoted by p_{ij}, is

$$p_{ij} = \frac{a_j v_j(\mathbf{x}_i)}{\sum_{j=1}^N a_j v_j(\mathbf{x}_i)}.$$

Therefore, a_j is the average number of points chosen from Gaussian j, weighted by $g_i = g(\mathbf{x}_i)$,

$$a_j = \sum_i g_i p_{ij},$$

$\mathbf{q}_j$ is the weighted average position of points drawn from Gaussian j,

$$\mathbf{q}_j = a_j^{-1} \sum_i g_i p_{ij} \mathbf{x}_i,$$

and M_j^{-1} the associated covariance matrix,

$$M_j^{-1} = a_j^{-1} \sum_i g_i p_{ij} \mathbf{x}_i \mathbf{x}_i^T - \mathbf{q}_j \mathbf{q}_j^T.$$

If $g_i = f_i$ for all i, then $\theta = \{a_j, \mathbf{q}_j, M_j\}$ is a fixed point of the map $\theta \to \theta' = \{a'_j, \mathbf{q}'_j, M'_j\}$ given by:

$$\begin{aligned} a'_j &= \sum_i f_i p_{ij}, \\ \mathbf{q}'_j &= a'^{-1}_j \sum_i f_i p_{ij} \mathbf{x}_i, \\ M'^{-1}_j &= a'^{-1}_j \sum_i f_i p_{ij} \mathbf{x}_i \mathbf{x}_i^T - \mathbf{q}'_j \mathbf{q}'^T_j. \end{aligned}$$

2.1.1 Implementation

I implemented the 1D case with f being a finite sum of N Gaussian functions. The algorithm works well, however, it gets often caught-up in a local maximum and does not improve after a certain point.

2.2 Tensorflow

To use more brute force, I implemented a tensorflow deep learning algorithm for computing the number of Gaussians N. That is, given a 1D f consisting of up to n Gaussian terms with a fixed variance and random mean, the program is taught to recognize their number. Tuning the batch, training and test size, using 3 hidden layers, the program was able to make a 0.97 correct prediction.