Neural Networks cheat-sheet

Andrea Jemmett

October 22, 2015

1 Perceptron

Perceptron Convergence Algorithm

For linearly sep. data; decision boundary: $\sum \mathbf{w_i x_i} + b = 0$

1. Variables & parameters:
   $\mathbf{x}(n) = [1, x_1(n), \ldots, x_m(n)]^T$
   $\mathbf{w}(n) = [b, w_1(n), \ldots, w_m(n)]^T$
   $y(n)$ = net out $\quad d(n)$ = target $\quad \eta$ = learning rate
2. *Initialization* $\mathbf{w}(0) = \mathbf{0}$ then for $n = 1, 2, \ldots$ do the following
3. *Activation* Feed input $\mathbf{x}(n)$ to network
4. *Compute actual response* as $y(n) = (\mathbf{w}^T(n)\mathbf{x}(n))$
5. *Adaptation of weight vector* update weights using:
   $\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$
   where:
   $d(n) = \begin{cases} +1 & \text{if } x(n) \text{ belongs to class } C_1 \\ -1 & \text{if } x(n) \text{ belongs to class } C_2 \end{cases}$

2 Statistic Based Methods

1. *Observation density / class-conditional / likelihood*
   $P(X = x|C_i) = \frac{\#\text{ x samples}}{\#\text{ of samples in } C_i}$
2. *Prior* $P(C_i) = \frac{\#\text{ samples in } C_i}{\#\text{ all samples}}$
3. *Posterior* $P(C_i|X = x) = \frac{\text{likelihood x prior}}{\text{evidence}}$
4. *Evidence* $P(X = x)$ is normalization / scaling factor

$$\mathbf{w_{MAP}} = \underset{\mathbf{w}}{\operatorname{argmax}}\, \pi(\mathbf{w}|d, \mathbf{x})$$

3 Linear Models

Gradient Descent Algorithm

1. Start from arbitrary point $\mathbf{w}(0)$
2. find a direction by means of a gradient: $\nabla\xi = [\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_n}]$
3. make a small step in that direction: $\Delta\mathbf{w} = -\eta\nabla\xi$
4. repeat the whole process

ADALINE uses an identity activation (continuous error measure) function and update rule is

$$\Delta\mathbf{w} = +\eta\mathbf{x}(d - y)$$

Linear regression uses sigmoid activation function and delta rule is

$$\Delta\mathbf{w} = +\eta(d - \varphi(net))\varphi'(net)\mathbf{x}$$

Cover's Theorem "A complex pattern-classification problem, cast in a high dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated."

4 Multi-Layer Perceptrons

Considering NN for XOR:

- Network output $y = \varphi(net) = \varphi(y_1u_1 + y_2u_2) = \varphi(u_1\varphi(net_1) + u_2\varphi(net_2))$
- Network error $e = \frac{1}{2}(d - y)^2$

Generalized Backprop delta rule

$$\Delta w_{ji} = \eta\delta_j y_i$$

$$\delta_j = \begin{cases} \varphi'(v_j)(d - y_j) & \text{if } j \text{ is output node} \\ \varphi'(v_j)\sum_k \delta_k w_{kj} & \text{if } j \text{ is hidden node} \end{cases}$$

5 Radial-Basis Function nets

Main idea: build local model of reference points and combine them.

- *Hidden layer* returns closeness from reference points
- *Output layer* standard linear regression (like ADALINE)
- *Closeness* is a radial function of the Euclidean distance:

$$\phi(||x - t_i||) \qquad \phi(r) = exp(-\tfrac{r^2}{2\sigma^2})$$
$$\phi(r) = (r^2 + \sigma^2)^{-\alpha}, \quad \alpha > 0 \qquad \phi(r) = r^2\ln(r)$$

Training:

1. Learn centres using K-means (unsupervised)
2. Learn weights from hidden to output using LMS (supervised)

6 Support Vector Machines

Main idea: **maximize margin around decision hyperplane**; decision function is specified by a subset of training samples: the support vectors.

$$\mathbf{w_o^T x_i} + b_o \geq +1 \qquad \text{when } d_i = +1 \qquad (1)$$
$$\mathbf{w_o^T x_i} + b_o \leq -1 \qquad \text{when } d_i = -1 \qquad (2)$$

Maximizing the margin of separation ρ is equivalent to minimize the Euclidean norm of the weight vector $\mathbf{w}$

$$\rho = \frac{2}{||\mathbf{w_o}||}$$

Problem. Find values of $\mathbf{w}$ that minimize

$$\phi(\mathbf{w}) = \frac{1}{2}\mathbf{w^T w}$$

given constraints

$$d_i(\mathbf{w^T x_i} + \mathbf{b}) \geq 1 \qquad \text{for } i = 1, 2, \ldots, N$$

Lagrangian function (linearly separable) ($\alpha_i > 0$ for support vectors)

$$J(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w^T w} - \sum_{i=1}^{N} \alpha_i [d_i(\mathbf{w^T w} + b) - 1]$$

Solution

$$Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=0}^{N} \alpha_i \alpha_j d_i d_j \mathbf{x_i^T x_j}$$

By setting partial derivatives to zero we obtain

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x_i}$$

Solution Non linear with kernel function $K(x_i, x_j)$

$$Q(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=0}^{N} \alpha_i \alpha_j d_i d_j \phi(\mathbf{x_i^T})\phi(\mathbf{x_j})$$

Possible **kernel functions**

- polynomial $K(x, y) = (xy + 1)^p$
- RBF gaussian $K(x, y) = \exp(-\frac{||x-y||^2}{2\sigma^2})$
- sigmoid $K(x, y) = \tanh(kxy - \delta)$

7 Principal Component Analysis

Main idea: **discover significant patterns or features in input data through use of unlabelled examples**.

Statistical method

Step 1 start with a dataset and subtract mean

Step 2 calculate covariance matrix

Step 3 calculate the eigenvectors and eigenvalues; the eigenvector with the highest eigenvalue is *the* principal component

Step 4 select how much to reduce the dimension of the dataset by selecting the p highest principal components

Self-organization

Self-amplification Hebb's postulate: If two neurons on either side of a connection are activated simultaneously (i.e. synchronously), then the strength of that connection is selectively increased. In the case of asynchronous activation the connection strength is decreased

Competition neurons compete with each other in a winner takes all fashion

Cooperation modification in synaptic weights and neurons tend to cooperate

Structural information redundant information is acquired in the form of knowledge

Hebbian learning rule

$$\Delta w_{ji} = \eta(y_i x_i - y_i \sum_{k=1}^{j} w_{ki} y_k)$$

8 Self-Organizing Maps

Goal: Transform an incoming signal pattern of arbitrary dimension into a one- or two-dimensional discrete map, and perform this translation adaptively in a topologically ordered fashion

Three processes:

1. **Competition** : find the winning neuron: $i(\mathbf{x}) = \text{argmin}_j \|\mathbf{x} - \mathbf{w_j}\|$
2. **Cooperation** : determine neighbourhood function: $h_{j,i} = \exp(-\frac{d_{j,i}^2}{2\sigma^2})$ where $d_{j,i}$ is the lateral distance from winning neuron
3. **Adaptation** : adapt weights with: $\mathbf{w_j}(n+1) = \mathbf{w_j}(n) + \eta(n)h_{j,i}(n)(\mathbf{x}(n) - \mathbf{w_j}(n))$

Two phases:

1. Ordering phase: topological ordering of the weight vectors; can take 1000 iterations with $\eta = [0.1, 0.01]$
2. Convergence phase: fine tune the map and provide accurate statistical quantification (small η)

Contextual maps Visualize SOM in a different way. *Feature map* see which neurons are excited when an unseen test pattern is presented. *Semantic map* creates a form of clustering or unsupervised categorization