Locally Weighted Regression

- KNN forms local approximation to f for each query point xq
- Why not form an explicit approximation f(x) for region surrounding xq

   Locally Weighted Regression

- **Locally weighted regression** uses nearby or distance-weighted training examples to form this local approximation to f. Its an instance based learning algorithm.
- We might approximate the target function in the neighborhood surrounding x, using a linear function, a quadratic function, a multilayer neural network.
- The phrase "locally weighted regression" is called
    - ***local*** because the function is approximated based only on data near the query point,
    - ***weighted*** because the contribution of each training example is weighted by its distance from the query point, and
    - ***regression*** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions.

Locally Weighted Regression

- Given a new query instance xq, the general approach in locally weighted regression is to construct an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding xq.
- This approximation is then used to calculate the value $\hat{f}$(xq), which is output as the estimated target value for the query instance.
- The description of $\hat{f}$ may then be deleted, because a different local approximation will be calculated for each distinct query instance.

Gradient Descent method for modifying weights

In this method error term is calculated by finding the square of the difference between target output and calculated output.

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

- η is the constant learning rate
- Weights are modified again and again until we get the minimum error.
- Choose the weights that minimizes the squared error summed over the training examples using Gradient Descent.

- The target function f is approximated near xq using a linear function of the form

$$\hat{f}(x) = w_0 + w_1 a_1(x) + \ldots\ldots + w_n a_n(x)$$

$a_i(x)$ denotes the value of the ith attribute of the instance x. Gradient descent method is used to find the coefficients $w_0 \ldots\ldots w_n$

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

$$\Delta w_j = \eta \sum_{x \in D} (f(x) - \hat{f}(x)) a_j(x)$$

But this is global approximation approach. It should be converted into local approach.

There is a need to modify this procedure to derive a local approximation rather than a global one. The simple way is to redefine the error criterion E to emphasize fitting the local training examples. Three possible criteria are given below

1. Minimize the squared error over just the k nearest neighbors:

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in \text{ k nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2$$

2. Minimize the squared error over the entire set D of training examples, while weighting the error of each training example by some decreasing function K of its distance from xq:

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

3. Now combine 1 and 2 to get locally as well as weighted regression

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in \text{ k nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$

If we choose the criterion 3 and rederive the gradient decent rule, following equation can be derived

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x)) \, (f(x) - \hat{f}(x)) \, a_j(x)$$

Now the contribution of instance x to the weight update is now multiplied by the distance penalty K(d(xq,x)), and that error is summed over only the k nearest training examples.

We can find modified weights. Start with random weights and modify them to get weights with minimum squared error.

Radial Basis Functions

- One approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks is learning with radial basis functions.
- A radial basis function network is a type of supervised artificial neural network that uses supervised machine learning (ML) to function as a nonlinear classifier.
- Radial basis function is used where data is not linearly separable. To make the data linearly separable it is horizontally expanded and vertically compressed. For this we are using Gaussian function.
- The learned hypothesis is a function of the form $$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

where each x_u is an instance from X and where the kernel function $K_u(d(x_u, x))$ is defined so that it decreases as the distance $d(x_u, x)$ increases.

- Even though $\hat{f}(x)$ is a global approximation to f(x), the contribution from each of the $K_u(d(x_u, x))$ terms is localized to a region nearby the point x_u.

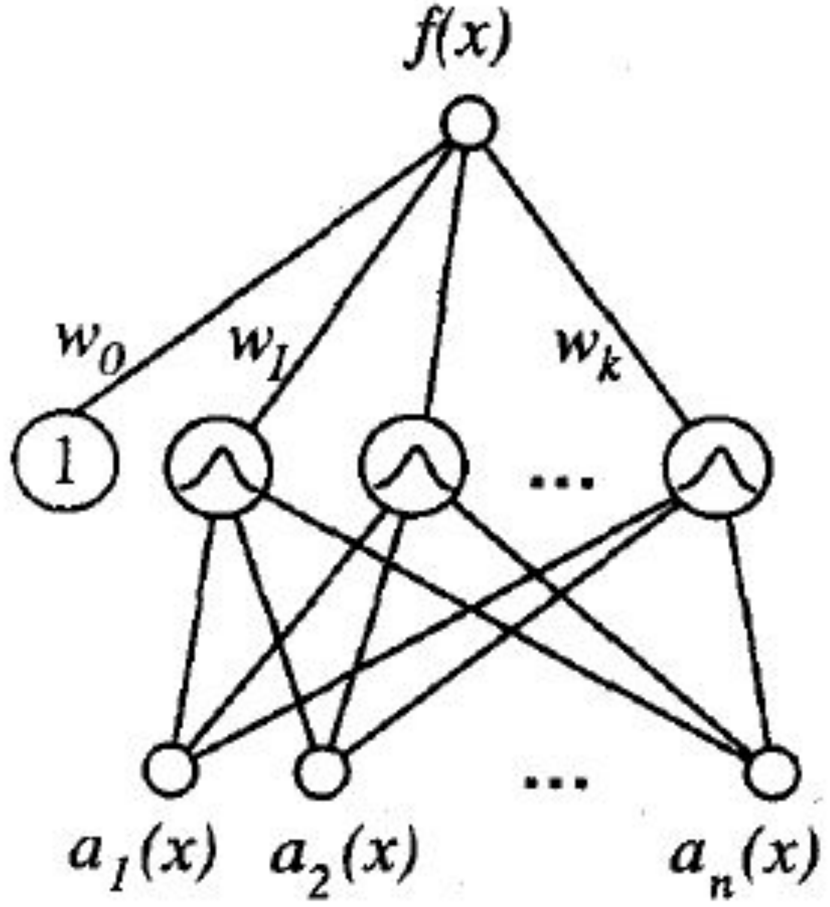
$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

Examples of radial basis functions

- linear radial basis function $\phi(r)=r$
- Multiquadratic radial basis functi $\phi(r) = \sqrt{r^2 + c^2}$
- Gaussian kernel $\phi(r) = \exp(-c^2 r^2)$
- Inverse multiquadratic radial basis func $\phi(r) = 1/\sqrt{r^2 + c^2}$

  where scalar parameter c which may be adjusted to improve the approximation

Radial Basis Function Networks



$$f(x) = w_0 + \sum_{u=1}^{k} w_u K_u(d(x_u, x))$$

$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

Each hidden unit produces an activation determined by a Gaussian function centered at some instance *xu.*

Therefore, its activation will be close to zero unless the input x is near *xu.*

The output unit produces a linear combination of the hidden unit activations.

Case-based Reasoning

- Instance-based methods
  - lazy
  - classification based on classifications of near (similar) instances
  - data: points in n-dim. space
- Case-based reasoning
  - as above, but data representted in symbolic form
- New distance metrics required

Case Based Reasoning (CBR)

Case-Based Reasoning classifiers (CBR) use a database of problem solutions to solve new problems

It stores the tuples or cases for problem-solving as complex symbolic descriptions.

How CBR works?

When a new case arises to classify, a Case-based Reasoner(CBR) will first check if an identical training case exists. If one is found, then the accompanying solution to that case is returned. If no identical case is found, then the CBR will search for training cases having components that are similar to those of the new case.

Conceptually, these training cases may be considered as neighbours of the new case.
The CBR tries to combine the solutions of the neighbouring training cases to propose a solution for the new case.

Four step process for CBR

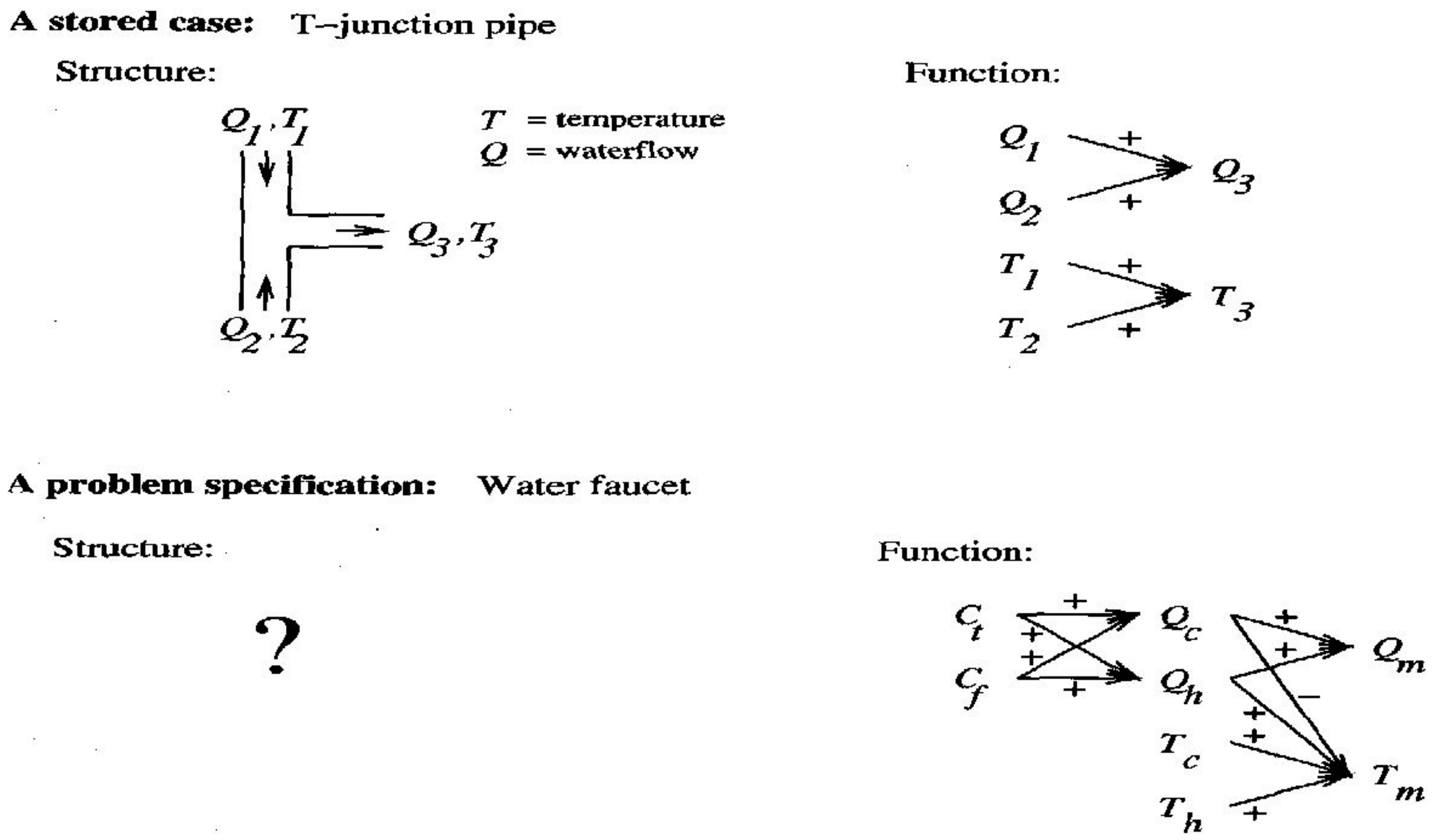
In general, the case-based reasoning process entails:

1. Retrieve- Gathering from memory an experience closest to the current problem.
2. Reuse- Suggesting a solution based on the experience and adapting it to meet the demands of the new situation.
3. Revise- Evaluating the use of the solution in the new context.
4. Retain- Storing this new problem-solving method in the memory system.

The **CADET system** employs case based reasoning to assist in the conceptual design of simple mechanical devices such as water faucets.

It uses a library containing approximately 75 previous designs and design fragments to suggest conceptual designs to meet the specifications of new design problems. Each instance stored in memory is represented by describing both its structure and its qualitative function.

New design problems are then presented by specifying the desired function and requesting the corresponding structure.


A stored case: T–junction pipe
Structure:
Function:
Q_1, T_1
Q_2, T_2
Q_3, T_3
T = temperature
Q = waterflow
Q_1
Q_2
Q_3
T_1
T_2
T_3
+
A problem specification: Water faucet
Structure:
?
Function:
C_t
C_f
Q_c
Q_h
T_c
T_h
Q_m
T_m
+
–