1.- ALGORITMOS RÁPIDOS PARA LA EJECUCIÓN DE FILTROS DE PILA

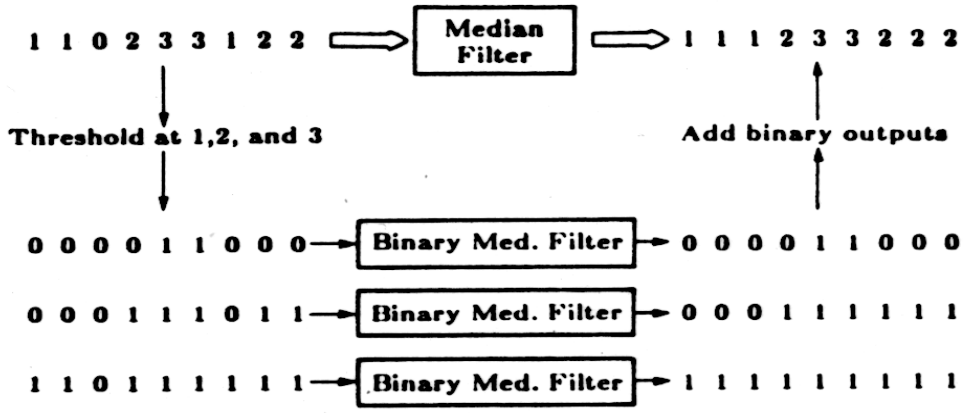
1.1.- INTRODUCCIÓN

Los filtros de pila constituyen una clase de filtros digitales no lineales. Un filtro de pila que es usado comúnmente para eliminar ruido mientras se preserva los bordes en la señal es el filtro mediana. Para un filtro mediana con ventana de longitud tres, su salida y_k , donde *k* denota el índice de tiempo, puede ser expresada en términos de la secuencia de entrada x_k como sigue:

$$y_k = max[min[x_{k-1}, x_k], min[x_k, x_{k+1}], min[x_{k+1}, x_{k-1}]] \qquad (1)$$

En general, todo filtro de pila opera eligiendo primero un número de subconjuntos de los puntos en la ventana, determinando el valor mínimo en cada uno de esos subconjuntos, y entonces usando el máximo de todos esos mínimos como la salida. Las maneras posibles de elegir esos subconjuntos se realiza por combinatoria y de hecho, hay más de $2^{2^{b/2}}$ filtros de pila de ventana con tamaño b.

Los filtros de pila poseen una débil propiedad de superposición llamada como *descomposición por umbral*. Como resultado, los filtros de pila pueden ser implementados en una arquitectura de descomposición por umbral, como se muestra a continuación:



En esta arquitectura, la señal de salida es descompuesta por umbral en una “pila” de señales binarias. La operación de filtrado es aplicada a cada una de las señales binarias en

paralelo con el uso de la lógica de Boole. La salida del filtro es la suma de la salida binaria sobre cada nivel. Para representar simples implementaciones de filtros de pila, la propiedad de descomposición por umbral facilita el desarrollo de metodología de diseño de filtros para filtros de pila (características estadísticos y estructural). Los filtros de pila designados usando esas dos características han sido mostrados y son muy efectivos en la supresión de ruido impulsivo y la simultánea preservación de los detalles en la imagen. Otras aplicaciones de los filtros de pila incluyen interpolación de señales, detección de bordes y memoria asociativa.

Algunos estudios recientes en el filtrado no lineal consideran el uso de lógica por umbral en la arquitectura de descomposición por umbral. La consecuencia viene por la búsqueda de una subclase útil de filtros de pila que sean efectivos en aplicación y simple en implementación. Esta clase de filtros son llamados filtros estadísticos de orden ponderado (WOS) y se utilizará bastante en los posteriores algoritmos.

1.2.- FILTROS DE PILA

Sea *R(n)* el proceso tiempo discreto a la entrada de un filtro de pila. Asumiremos que *R(n)* toma valores enteros en Q = {0, 1, 2,..., M -1}. Una ventana de tamaño b, donde b es algún entero impar, se desplaza a través de la entrada de proceso *R(n)*. Sea $\vec{R}(n)$ el vector conteniendo las b muestras de la ventana en un tiempo n.

$$\vec{R}(n) = \left[R\left(n - \frac{b-1}{2}\right) \cdots R(n) \cdots R\left(n + \frac{b+1}{2}\right) \right] \qquad (2)$$

A cada tiempo de instante n, el filtro de pila $S_f(\cdot)$ mapea $\vec{R}(n)$ a algún entero en Q. El mapeado $S_f(\cdot) : Q^b \longrightarrow Q$ se define en términos de la arquitectura de descomposición por umbral mostrada en la fig. 1.

$$S_f\left(\vec{R}(n)\right) = \sum_{k=1}^{M-1} f\left(T_k\left(\vec{R}(n)\right)\right) \qquad (3)$$

donde

$$T_k\left(\vec{R}(n)\right)=\left[T_k\left(R\left(n-\frac{b-1}{2}\right)\right)\cdots T_k\left(R(n)\right)\cdots T_k\left(R\left(n+\frac{b+1}{2}\right)\right)\right] \quad (4)$$

en la cual

$$T_k\left(R(n)\right)=\begin{cases}1 & si \quad R(n)\geq k\\ 0 & cualquier \quad otro\end{cases} \quad (5)$$

y $f(\cdot)$ es la función booleana usada en cada nivel de la arquitectura. Esto es, la señal de entrada es descompuesta por umbral en M - 1 señales binarias, cada una de las señales binarias es filtrada por tablas booleanas. La salida es la suma de las salidas binarias de todos los niveles. Nótese que la función booleana en cada nivel está sujeta a una condición de consistencia llamada como la propiedad de pila. Este propiedad requiere la salida sobre un nivel *l* que debería ser mayor o igual a la salida en un nivel *l +1*. Matemáticamente, impondremos la siguiente condición:

$$f\left(\vec{\alpha}_i\right)\leq f\left(\vec{\alpha}_j\right) \qquad cuando \qquad \vec{\alpha}_i\leq\vec{\alpha}_j \quad (6)$$

para cualquiera de las dos longitudes binarias *b* en las secuencias $\vec{\alpha}_i,\vec{\alpha}_j$. Aquí, $\vec{\alpha}_i\leq\vec{\alpha}_j$ implica que cada entrada de $\vec{\alpha}_i$ es menor o igual que la correspondiente entrada en $\vec{\alpha}_j$. Por ejemplo, $(010)\leq(110)$ y la propiedad de pila dicta que $f(010)\leq f(110)$.

1.3.- EJECUCIÓN RÁPIDA DE ALGORITMOS PARA FILTROS DE PILA

Aquí, para encontrar un óptimo filtro de pila bajo el significado del criterio de error absoluto dado por

$$C=E\left[S(n)-\sum_{k=1}^{M-1}f\left(T_k(\vec{R}(n))\right)\right] \quad (7)$$

donde $S(n)$ es el proceso deseado, se podrá solucionar por un programa lineal. El algoritmo sería:

Algoritmo Lin's

inicio

Desde n = 1 **hasta** n = N **hacer**

Desde k = 1 **hasta** k = M -1 **hacer**

1. Calcula el par de umbral $\left(T_k\left(\vec{R}(n)\right), T_k\left(S(n)\right)\right)$ y actualiza la tabla booleana a la entrada $T_k\left(\vec{R}(n)\right)$.

2. Aplica la propiedad de pila.

fin_desde

Convierte la tabla booleana de débil a fuerte

fin_desde

Se ha diseñado un algoritmo similar a este pero mucho más rápido

Algoritmo rápido

inicio

Desde n = 1 **hasta** n = N **hacer**

Desde i = 1 **hasta** i = b + 2 **hacer**

1. Actualiza la tabla booleana a la entrada $T_{R_{(i)}(n)}\left(\vec{R}(n)\right)$ por la longitud del paso $\Delta_i(n)$

2. Aplica la propiedad de pila.

fin_desde

Convierte la tabla booleana de débil a fuerte

fin_desde

Similar al **Algoritmo Lin´s**, el **Algoritmo rápido** requiere solamente operaciones aritméticas (incremento, decremento y comparaciones locales).

Teorema 3.1: Sean las entradas de salida de la tabla suave booleana en un tiempo *n*, se representa como un vector de longitud 2^b y es denotada como $\vec{X}(n)$. La evolución de $\vec{X}(n)$ puede ser descrita por

$$\vec{X}(n+1) = h\left(\vec{X}(n), \vec{R}(n), \vec{S}(n)\right) \qquad (8)$$

donde $h(\cdot)$ denota el **Algoritmo rápido** visto anteriormente.

1.4.- ALGORITMOS DE EJECUCIÓN RÁPIDA PARA FILTROS WOS

Los filtros WOSA son una subclase de filtros de pila donde la lógica por umbral se usa en cada nivel en la arquitectura de descomposición por umbral. Una lógica de umbral es una función booleana donde la salida es determinada comparando la suma ponderada de sus bits de entrada con algún valor umbral. Específicamente, la salida de una lógica por umbral con el vector de ponderación $\vec{v}$ y umbral α para cualquier entrada $\vec{r}$ puede ser expresada como

$$f(\vec{r}) = \begin{cases} 1 & si \quad \vec{v}^t\vec{r} \geq \alpha \\ 0 & cualquier \quad otro \end{cases} \qquad (9)$$

donde el superíndice *t* significa transpuesta y α es un escalar. La ventaja de la lógica por umbral es que es simple de implementar para largas longitudes de ventanas y sólo el peso y el umbral necesitan ser almacenados.

Cuando una lógica por umbral es usada en una arquitectura de descomposición por umbral, la salida de un filtro de pila para un tiempo *n* es

$$S_f\left(\vec{R}(n)\right) = \sum_{k=1}^{M-1} T_\alpha\left(\vec{v}^t\vec{r}_k(n)\right) \qquad (10)$$

donde

$$\vec{r}_k(n) = 2T_k\left(\vec{R}(n)\right) - \vec{1}. \qquad (11)$$

Aquí, la representación simétrica es adoptada por la entrada binaria en la lógica por umbral. Nótese que si todos las ponderaciones son restringidas para ser no negativas, entonces la propiedad de pila se mantiene. Al menos que esta condición sea suficiente pero no necesaria, se adoptará en este estudio por simplicidad.

La ponderación y el umbral deben ser elegidos lo mejor posible para minimizar

$$C = E\left[\sum_{k=1}^{M-1}\left(d_k(n) - \vec{v}^{\,t}\vec{r}_k(n) - \alpha\right)^2\right] \qquad (12)$$

donde

$$d_k(n) = 2T_k(S(n)) - 1. \qquad (13)$$

En la práctica, es conveniente mantener el umbral α como una de las ponderaciones a ser ejecutadas y su correspondiente bit de entrada es siempre igual a -1, por ejemplo, sea

$$\vec{x}_k(n) = (\vec{r}_k(n), -1)$$

(14)

$$\vec{w} = (\vec{v}, \alpha)$$

entonces C es equivalente a

$$C = E\left[\sum_{k=1}^{M-1}\left(d_k(n) - \vec{w}^{\,t}\vec{x}_k(n)\right)^2\right] \qquad (15)$$

Claramente, el $\vec{w}$ que minimiza (15) puede ser encontrado usando algoritmos de adaptación en filtros lineales, como los algoritmos LMS y RLS (véase más adelante). El procedimiento actualizado en esos algoritmos es similar a un algoritmo de adaptación para filtros de pila. En un tiempo *n*, cada uno de los umbrales M - 1 de par $d_k(n)$, $x_k(n)$, $k = 1, 2, M-1$, es

alimentado al algoritmo de adaptación secuencialmente para actualizar las ponderaciones. Al final del periodo de desplazado, los pesos negativos se sitúan a cero para mantener la propiedad de pila. En lo sucesivo, mostramos las consecutivas actualizaciones como la misma entrada que puede ser reducida a una simple actualización en los algoritmos LMS y RLS.

A. Algoritmo rápido basado en LMS.

Sea $\vec{w}(m), \vec{x}(m)$ las ponderaciones y las entradas en ejecución en el algoritmo LMS para un tiempo m (o de forma más precisa, en la iteración *m* bajo el criterio de error dado en (16) y entonces $\vec{w}(m)$ es actualizado como sigue:

$$\vec{w}(m) = \vec{w}(m-1) + \mu\, c(m|m-1)\vec{x}(m) \qquad (16)$$

donde μ es el tamaño de paso y

$$e(m|m-1) = d(m) - \vec{w}^t(m-1)\vec{x}(m) \qquad (17)$$

es el error de predicción $d(m)$ usando $\vec{w}(m-1)$ y $\vec{x}(m)$.

Supongamos en las interacciones desde m hasta $m+\Delta-1$las entradas en el algoritmo son repetitivas y por conveniencia se denotan como $\vec{x}$ y d. Entonces, $e(l|l-1)$ tiene la siguiente relación recursiva

$$\begin{aligned} e(l|l-1) &= d - \vec{w}^t(l-1)\vec{x} = d - \left(\vec{w}(l-2) + \mu\, e(l-1|l-2)\vec{x}\right)^t \vec{x} = \\ &= e(l-1|l-2)\left(1 - \mu\,\|\vec{x}\|^2\right) \qquad para \qquad m+1 \le l \le m+\Delta-1. \end{aligned} \qquad (18)$$

Aplicando (18) recursivamente, tendremos

$$e(l|l-1) = e(m|m-1)\left(1 - \mu(b+1)\right)^{(l-m)} \qquad (19)$$

donde $\|\vec{x}\|^2 = b+1$ debido a la representación simétrica en $\vec{x}$. Aplicando (16) recursivamente, tendremos para $m \le l \le m+\Delta-1$

$$\vec{w}(l) = \vec{w}(l-1) + \mu\, e(l|l-1)\vec{x} = \vec{w}(l-2) + \mu\, e(l-1|l-2)\vec{x} + \mu\ e(l|l-1)\vec{x} = \\ = \cdots = \vec{w}(m-1) + \mu \sum_{j=m}^{l} e(j|j-1)\vec{x} \tag{20}$$

Sustituyendo (20) en la ec. superior y reemplazando l por $m+\Delta-1$

$$\vec{w}(m+\Delta-1) = \vec{w}(m-1) + \mu\, e(m|m-1) * \left[\sum_{j=m}^{m+\Delta-1} \left(1-\mu(b+1)\right)^{(j-m)}\right]\vec{x} = \\ = \vec{w}(m-1) + e(m|m-1)\frac{1-\left(1-\mu(b+1)\right)^{\Delta}}{b+1}\vec{x} \tag{21}$$

Las funciones de arriba indican que si la entrada en ejecución del algoritmo LMS es repetitiva sobre un periodo de tiempo $[m, m+\Delta-1]$, entonces ésas actualizaciones Δ del vector de ponderación acorde con (16) pueden ser hechas en un paso como en (21). Las ecuaciones (21) y (16) son similares salvo en el "tamaño del paso". Para el último, es $\frac{1-\left(1-\mu(b+1)\right)^{\Delta}}{b+1}$, que es aproximadamente igual $\Delta \times \mu$ para pequeños μ.

Éste sería el listado del algoritmo rápido basado en LMS para filtros WOS:

Algoritmo rápido basado en LMS

inicio

Desde n = 1 **hasta** n = N **hacer**

Desde i = 1 **hasta** i = b + 2 **hacer**

1. Halla el par de umbral $(\vec{x}, d)$ y el tamaño del paso Δ
2. Halla el error de predicción

$$e = d - \vec{w}_{old} \cdot \vec{x}$$

3. Actualiza las ponderaciones $\vec{w}$ de acuerdo con (21):

$$\vec{w}_{new} = \vec{w}_{old} + e\frac{1-(1-\mu(b+1))^{\Delta}}{b+1}\vec{x}$$

fin_desde

El conjunto de ponderaciones negativas, se pasan a cero.

fin_desde

B.Algoritmo r basado en RLS.

El Algoritmo RLS encuentra el vector de ponderación RLS para minimizar la suma del actual error cuadrático

$$C = \sum_{n=1}^{N}\sum_{k=1}^{M-1}\left(d_k(n) - \vec{w}^t\vec{x}_k(n)\right)^2$$

donde N es la longitud de los datos en ejecución.

El listado sería el siguiente:

Algoritmo RLS

Inicialización

En m = 0:

$\vec{w}(0) = \vec{x}(0) = \vec{0}$

$C(0) = \delta\ I(\delta >> 1)$

inicio

Desde m = 1 **hasta** m = N **hacer**

1. Obtener $d(m)$ $\vec{x}(m)$

2. Hallar error de predicción:

$$e(m|m-1) = d(m) - \vec{w}^t(m-1)\vec{x}(m)$$

3. Hallar nuevo vector de ganancia:

$$\mu(m) = \vec{x}^t(m)C(m-1)\vec{x}(m)$$

$$\vec{g}(m) = \frac{C(m-1)\vec{x}(m)}{1+\mu(m)}$$

4. Actualizar el vector de ponderación:

$$\vec{w}(m) = \vec{w}(m-1) + \vec{g}(m)e(m|m-1)$$

5. Actualizar la matriz inversa C:

$$C(m) = C(m-1) - \vec{g}(m)\vec{x}^t(m)C(m-1).$$

fin_desde

Nuevamente, supongamos en el período de tiempo $[m, m+\Delta-1]$ las entradas en el algoritmo son las mismas y están denotadas como $\vec{x}$ y d. Asumimos que el índice l está dentro de $[m, m+\Delta-1]$ al menos que no se indique lo contrario y por tanto, simplificando tendremos

$$e(l|l-1) = d - \vec{w}^t(l-1)\vec{x}$$