MONOGRAFÍA

TEMA:

PSQA: MEDIDA DE CALIDAD DE VIDEO EN TIEMPO REAL

Asignatura: Int. Al Reconocimiento de Patrones
Integrantes: Pablo Romero y Miguel Tasende

INTRODUCCIÓN

El tema central de esta monografía es PSQA (por sus siglas en inglés "Pseudo Subjective Quality Assessment"), como técnica para medir la calidad de video en tiempo real sobre redes de paquetes. El nexo con la asignatura es estrecho, fundamentalmente mediante el uso de redes neuronales aleatorias (de ahora en más RNN, por su sigla en inglés).

Este material está dirigido a estudiantes avanzados en ingeniería eléctrica e ingeniería en computación, y más en general a aquellos que deseen iniciar en las distintas técnicas que existen a la hora de medir calidad de video.

El documento se ordena de la siguiente manera. En el capítulo 1 brindamos un marco teórico, en el que se muestran los elementos de la calidad de video (centrado en PSQA) y RNN. El capítulo 2 muestra los resultados obtenidos en este trabajo, detallando el proceso de entrenamiento de las RNN. Finalmente se presenta una referncia bibliográfica y un apéndice con todos los programas realizados en Matlab.

Deseamos que disfruten su lectura, así como los autores tanto en el proceso de aprendizaje como en la realización de la misma.

AGRADECIMIENTOS

Este trabajo sencillamente hubiera sido imposible sin la cooperación de las personas que mencionamos a continuación:

Al Dr. Pablo Rodríguez Bocca, que con gran amabilidad nos ha brindado materiales valiosísimos e imprescindibles para este trabajo, como también nos ha respondido al instante cuestiones referentes a nuestro problema. Al Dr. Franco Robledo, quien nos motiva constantemente en problemas de redes de pares y sistemas eficientes de distribución de video, y nos permitió conocer PSQA y la problemática actual de distribución de video en redes de pares. Al Lic. Sebastián Basterrech, que también nos brindó importante material de consulta y nos sugirió una manera de trabajar, puesto que él ya tiene más tiempo que nosotros en medidas de calidad de video. Tenemos muy presente y agradecemos también la atención del Dr. Héctor Cancela, y el placer de enseñar de nuestros docentes de la asignatura Int. al Reconocimiento de Patrones, que deseamos sigan dando sus frutos.

Indice

MARCO TEORICO

INTRODUCCION

Este capítulo contiene 2 secciones. La primera refiere a elementos en la calidad del video. Se resalta la técnica pseudosubjetiva PSQA con respecto a las anteriores técnicas de evaluación subjetiva y objetiva de video, debido básicamente al requisito de evaluación de la calidad del mismo en tiempo real y a su bajo costo de aplicación, entre otras que se detallarán.

Al aplicar PSQA surge la necesidad de aproximar una función que mide la calidad. Procedemos aquí entrenando una red neuronal aleatoria, o RNN.

El capítulo 2 es un resumen de RNN, que se basa en una fusión de ideas entre las tradicionales redes neuronales artificiales (ANN) y teoría de colas. Nos centraremos en la red neuronal aleatoria de 3 capas (una capa oculta), en la que existen numerosos trabajos previos con buenos resultados, es decir, logran hallar una función que es similar a la calidad subjetiva del video, en un sentido de bajo error medio cuadrático.

Finalmente en este capítulo mostramos la técnica utilizada a los efectos de entrenar nuestra red, que se basa en un descenso de gradiente. Detallaremos la expresión del vector gradiente para el caso de la red de 3 capas, y la metodología de descenso y actualización de los pesos.

SECCION 1. CALIDAD DEL VIDEO

Medidas objetivas y subjetivas de la calidad del video

Históricamente existen 2 técnicas de evaluación de la calidad de video, a saber: objetivas y subjetivas.

Las primeras definen una medida de la calidad del video distorsionado en base a una comparación con el original. En una red de paquetes usualmente toman como parámetros de calidad del video la tasa de bits (BR), tasa de pérdidas (LR), retardo y fluctuación, nivel de cuantización en la compresión y otros. Se adaptan bien a la hora de maximizar la calidad del servicio (QoS) entregado en una red de paquetes, pero muy pocas veces a la calidad de experiencia (QoE) percibida por los usuarios (según medidas subjetivas). Por otro lado, en general requieren acceder tanto al video original como al distorsionado simultáneamente para

declarar una calidad al video, por lo que no es posible una medición en tiempo real.

Algunos ejemplos de medida objetiva son PSNR (Peak SNR), VQM (Video Quality Metric), MPQM (Moving Picture Quality Metric) y CMPQM (Color MPQM) y NVFM (Normalization Video Fidelity Metric).

Varían en la complejidad, como también el nivel de correlación con la calidad subjetiva. PSNR por ejemplo es muy sencillo (tan solo computa una suma de las diferencias entre niveles de luminancia a nivel de píxeles), pero está lejos de medir calidad de manera similar a los humanos. MPQM y su extensión con agregado de color CMPQM presentan mayor complejidad, y correlacionan bien con medidas subjetivas, al menos en escenarios particulares (con BR altos).

Por otra parte, las técnicas subjetivas definen la calidad en base a la opinión de múltiples usuarios, que juzgan la calidad de muestras de video distorsionadas. Es la mejor manera de determinar la calidad de experiencia de los usuarios. Sin embargo, presenta serias dificultades de aplicación. La primera es que por definición no son posibles de realizar en tiempo real. La segunda es que su realización requiere disponer de recursos humanos, y consume mucho tiempo y esfuerzo.

Existen medidas subjetivas tanto cualitativas como cuantitativas. Las cualitativas no se trasladan bien a escalas numéricas, aunque pueden ser útiles a la hora de negociar un contrato de servicio de entrega de video.

En las medidas subjetivas cuantitativas, un grupo de personas juzgan la calidad del video con una puntuación numérica, y son más apropiadas para la medición de calidad de experiencia. La medida se brinda comúnmente a partir de un promediado de puntuaciones, conocido como MOS (Mean Opinion Score). El estándar ITU-R BT.500-11 detalla métodos para efectuar medidas subjetivas cuantitativas de calidad.

DCR (Degraded Category Rating)

El método subjetivo de evaluación DCR es también conocido como DSIS (Double Stimulus Inpairment Scale), y se especifica en el estándar ITU-R BT.500-11.

Las muestras de video se transmiten de a pares: en primera instancia el video original e inmediatamente después (no más de 10 segundos de espera) el video distorsionado, el cual debe ser puntuado por el panel según una escala del 1 al 5 que se muestra en la figura 1. El tiempo total de una sesión para cada miembro del panel no debe superar la media hora para no agregar el efecto del cansancio, elemento a ser tomado en cuenta para determinar los días necesarios para tener los resultados finales.

El resultado final va a ser un promedio o MOS para cada una de las configuraciones de video distorsionadas. Vale destacar que frecuentemente existe un proceso intermedio para eliminar “malas muestras”, consideradas como tales cuando se apartan de la media de opiniones. El MOS se vuelve a medir en borrando estas últimas muestras.

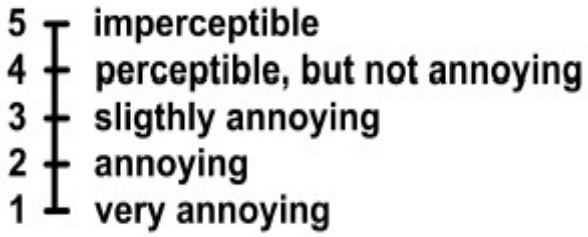


Fig. 1: escala usada en DCR

Se observa que a los efectos de medir la calidad de experiencia del video en tiempo real no son aptas las estrategias clásicas de evaluación objetiva ni subjetiva.

Las dos críticas a las objetivas son la baja correlación con la calidad percibid con los usuarios, e imposibilidad de obtener medidas de calidad en tiempo real, debido a la necesidad de poseer ambas muestras de video a la vez en un mismo extremo físico de la red.

Por otro lado, las técnicas subjetivas son el indicador de la calidad de experiencia de los usuarios. Sin embargo, por definición no son posibles de realizar en tiempo real, y además consumen tiempo y recurso humano, no siendo posible realizarlas con alta frecuencia.

Se propone entonces una nueva técnica pseudosubjetiva de evaluación de calidad de video, que no requiere las muestras distorsionada y original del video, y además es económica y fácil de automatizar, siendo posiblemente aplicable en tiempo real.

Otro enfoque: PSQA

PSQA (Pseudo Subjective Quality Assessment) es la técnica que nos va a permitir cumplir con nuestros objetivos. La idea básica consiste en obtener en primera instancia una medida subjetiva de la calidad del video para una cantidad finita de configuraciones distorsionadas para luego extrapolar esta función discreta entrenando una red neuronal aleatoria.

La etapa final es la evaluación pseudosubjetiva propiamente dicha, y sencillamente evalúa la función de calidad dados los parámetros de impacto del video distorsionado, que se deben medir.

Más específicamente, el procedimiento para efectuar PSQA consiste en los siguientes pasos

1- Determinar un conjunto de parámetros de impacto en la calidad de video
2- Construir un testbed con el que sea posible transmitir secuencias de video distorsionadas controladamente, eligiendo los valores de los parámetros de impacto antes seleccionados.
3- Elegir una cantidad razonable de configuraciones de video distorsionadas y un panel humano, para realizar una encuesta subjetiva.
4- Eliminar muestras malas, que son aquellas que difieren significativamente de la opinión media.
5- Calcular el MOS en base a las muestras buenas. Estos valores determinan para cada configuración distorsionada, la medida de calidad. Obsérvese que es tan solo un conjunto finito de valores. Deseamos medir la calidad variando los parámetros en un continuo.
6- Entrenar una red neuronal aleatoria. Esta red tiene como entrada los parámetros de impacto normalizados, y como salida un número, que debe ser similar al MOS (tanto para las muestras de entrenamiento como las de prueba).
7- Efectuar la medida pseudosubjetiva propiamente dicha de la calidad mientras fluye en una red de paquetes un streaming de video, midiendo los parámetros de impacto y evaluando la función antes obtenida.

Las figuras 2 y 3 ilustran la etapa previa a la medición (o de “entrenamiento” de la red neuronal) y la de utilización propiamente dicha. Nótese que una vez realizado el entrenamiento, si los parámetros de impacto siguen capturando la esencia de la calidad de experiencia del video, la misma red entrenada puede seguir siendo utilizada, y el proceso de encuestado y obtención de MOS lo realizamos tan solo una vez.

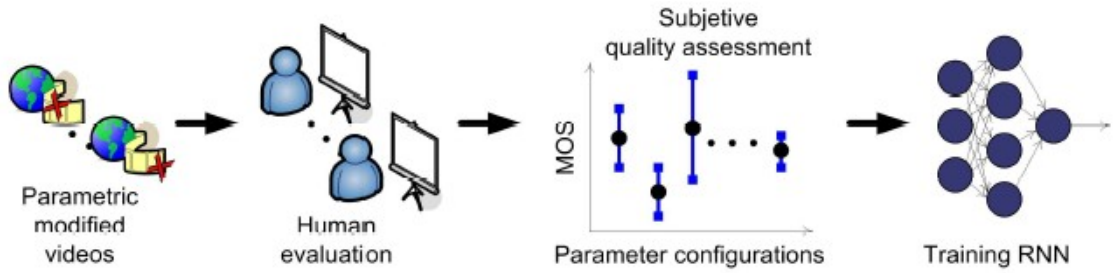


Fig. 3: Etapa de entrenamiento de PSQA

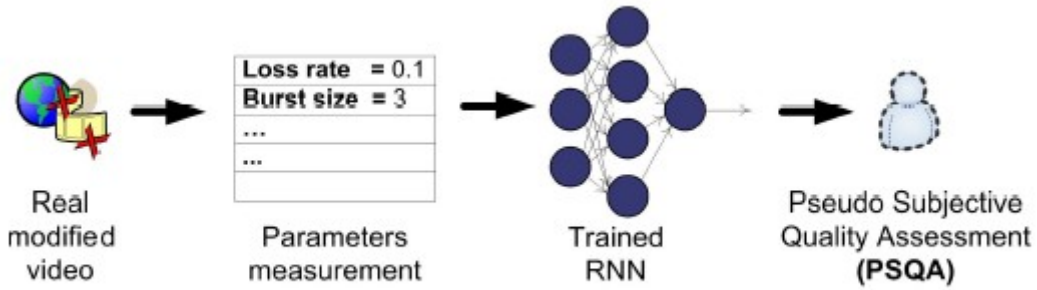


Fig. 4: Etapa de utilización de PSQA

Nuestro trabajo parte de datos ya obtenidos en una encuesta real puntuada por 5 expertos para 100 configuraciones diferentes de video variando 2 parámetros de impacto.

Parte central de nuestro trabajo es entonces el diseño y entrenamiento de una red neuronal aleatoria que extrapole esos valores, con previo filtrado (haciendo uso de la función filtro.m que se encuentra en el apartado de archivos en Matlab).

A continuación mencionamos brevemente los puntos del 1 al 5 del procedimiento de PSQA, concentrándonos finalmente en el punto 6 en el capítulo 2, que detalla el marco teórico de redes neuronales aleatorias y muestra elementos de diseño de redes con una capa oculta.

Paso 1: Parámetros de impacto

En la tesis de doctorado en la cual basamos nuestro trabajo, se han elegido parámetros orientados a medir la QoE en una red de pares. Un elemento innovador en esta tesis es basarse en pérdidas de cuadros de video, a diferencia de otros trabajos, en los que se mide la pérdida de paquetes, u otros que refieren a parámetros de calidad de la red.

La metodología de medición es a nivel de aplicación, dada la facilidad de agregado de software a nivel de usuario, inherente a las redes de pares.

Los parámetros seleccionados fueron 2: la tasa de pérdidas de cuadros, o LR (Loss Rate) y el número medio de cuadros perdidos en una ráfaga, o MLBS (Mean Loss Burst Size).

Paso 2: Control de la distorsión

Si bien podríamos simular un LR del 10 por ciento sencillamente decimando cuadros de video en forma determinística, esto no ocurre así en una red de paquetes real; las pérdidas son impredecibles. Existe un valioso modelo a ser considerado aquí para distorsionar la secuencia de video en forma estocástica, conocido por su autor como el modelo de Gilbert.

Consideremos la cadena de Markov discreta en el tiempo de 2 estados que se muestra en la figura 5. Es claro que el tiempo de permanencia de un estado se distribuye geométricamente, y su valor medio es el inverso de la probabilidad de transición. Además en estado estacionario, las probabilidades de los estados se hallan directamente a partir de la ecuación de equilibrio $\pi P = \pi$ sujeta a que $\pi = (\pi_0, \pi_1)$ es medida de probabilidad, donde P es la matriz de transición.

$$\pi_0 = \frac{p}{p+q}, \quad \pi_1 = \frac{q}{p+q}$$

Llevamos ahora esta simple cadena al contexto del video. Podemos hablar del estado 0 como "cuadro perdido" y el estado 1 como "transmisión correcta". Entonces, la esperanza del tiempo de permanencia del estado 0 es exactamente el MLBS, y la probabilidad del estado 1 en estado estacionario es igual al LR:

$$MLBS = \frac{1}{q}, \quad LR = \frac{q}{p+q}$$

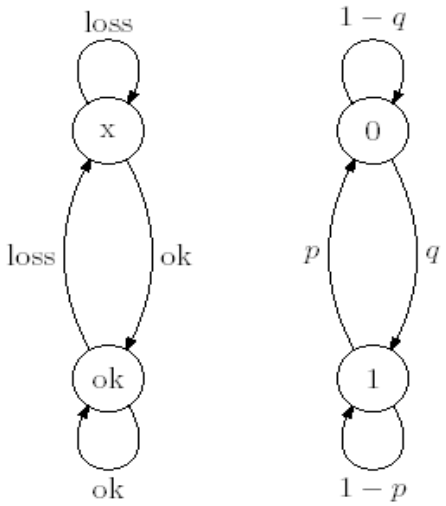


Fig. 5: Modelo de pérdidas de Gilbert y cadena de Markov de 2 estados

Pero entonces podemos emular las pérdidas de los cuadros de video a partir de una cadena de 2 estados, definiendo los parámetros de transición tales que cumplan con las anteriores igualdades:

$$q = \frac{1}{MLBS}, \quad p = \frac{1}{MLBS}\left(\frac{LR}{1-LR}\right)$$

El comportamiento de este modelo es muy simple, y se puede apreciar en el archivo gilbert.m dentro de los archivos de Matlab.

Paso 3: Elementos para la encuesta

La encuesta real en la que se basa nuestro trabajo tuvo como panel a 5 expertos en calidad de video, y se consideraron 100 configuraciones distintas de videos distorsionados, a partir de 25 distorsiones a cada uno de 4 videos cortos en formato de compresión MPEG-2.

Las 100 configuraciones se eligieron al azar en intervalos donde existe mayor sensibilidad al ojo humano: a partir de un LR del 20% el humano puntuará con 1 (lo peor), más allá de esas tasas de pérdidas no vale la pena estudiar. Lo mismo para MLBS superiores a 10.

El programa rand100.m es muy simple y genera 100 pares al azar, donde es posible elegir el rango de variación de cada coordenada.

Paso 4: Eliminar muestras malas

Este paso está documentado en el mismo programa filtro.m que se encuentra en la sección de programas de Matlab. Sencillamente elimina todas las muestras de miembros que puntuaron con valores muy lejanos al MOS de cada configuración de video. Para los datos de la encuesta considerada en este trabajo, no se ha eliminado muestras de ninguno de los cinco expertos evaluadores

Paso 5: Nuevo valor MOS y función de calidad

En general, al eliminar muestras se debe recalcular el MOS de cada configuración de video. En este caso particular el MOS calculado en el paso previo se conserva, pues no se eliminaron muestras.

El paso siguiente va a consistir en hallar una función de calidad, que va a ser evaluada cada vez que se quiera determinar la calidad de video.

Vamos a detallar en el próximo capítulo el paso 6, pero previamente vale aclarar el requisito que debe cumplir esta función.

Consideremos las ternas $(LR_1, M_1, m_1), (LR_2, M_2, m_2), ...(LR_{100}, M_{100}, m_{100}),$ que son las 100 evaluaciones obtenidas de la encuesta.

La función de calidad debe cumplir los siguientes dos requisitos:

- $f(LR_i, M_i) \approx m_i \quad \forall i = 1..100,$
- Para pares de parámetros posibles y no considerados en la encuesta, la función debe retornar valores próximos a los que se obtendría si se realizara efectiva una encuesta subjetiva con tales valores

SECCION 2: RNN

Introducción

Las redes neuronales aleatorias se basan en el sistema nervioso biológico, y mezcla herramientas de la teoría de colas y las clásicas redes neuronales.

La naturaleza excita ciertas neuronas de la red según los estímulos de entrada (que en nuestro caso son los parámetros de impacto del video), y estas en consecuencia aumentan su potencial en una unidad. La neurona atiende este impulso excitatorio y lo retransmite a neuronas vecinas con las que está comunicada. El nuevo impulso puede ser inhibitorio o bien excitatorio. En el primer caso reduce el potencial de la última neurona en una unidad (si es excitada aumenta la misma cantidad).

¿Qué vínculo tiene esto anterior con la teoría de colas? Pues bien, las neuronas atienden excitaciones (o clientes) a una tasa exponencial, y los arribos de la naturaleza son también exponenciales. Cada neurona se puede ver como un sistema $M/M/1$, donde la tasa de arribo va a depender de la cantidad de neuronas que le arriben y sus tasas de atención.

El estado de la red es el vector de potenciales de las neuronas que lo constituyen. Se puede probar un resultado similar al de redes de Jackson, y la probabilidad de cada estado de la red se puede calcular a partir de la productoria de probabilidades de estado de cada neurona de esta red.

La figura 6 ilustra el proceso que sufre una neurona particular. Se puede apreciar las distintas excitaciones e inhibiciones de otras neuronas, tanto a la salida como a la entrada. También hay elementos nuevos aquí: p_{ij}^{+} representa la probabilidad de que un cliente serviciado se dirija luego de ser atendido en la neurona i a la neurona j, como una excitación, p_{ij}^{-} representa el mismo caso, excepto que es una inhibición (quita un cliente en la cola de la neurona j), y d_i es la probabilidad de que el cliente luego de ser serviciado por la neurona i abandone la red.

A partir de estas definiciones, para cada neurona i de la red se debe cumplir la siguiente igualdad:

$$\sum_j (p_{ij}^{+} + p_{ij}^{-} + d_i) = 1$$

Directo de la teoría de colas se obtiene que el factor de utilización (en este contexto, la probabilidad de excitación, o potencial no nulo) es el cociente entre la tasa de entrada y la de salida.

En una red neuronal aleatoria genérica tenemos que

$$\rho_i = \frac{\mu_i^+ + \sum_j w_{ji}^+ \rho_j}{\mu_i + \sum_j w_{ij}^- \rho_j},$$

donde $w_{ij}^+ = \lambda_i p_{ij}^+$, $w_{ij}^- = \lambda_i p_{ij}^-$ representan las tasas efectivas entre las neuronas i y j (saliente de i y entrante a j). Juegan un rol similar a los pesos de una clásica red neuronal.

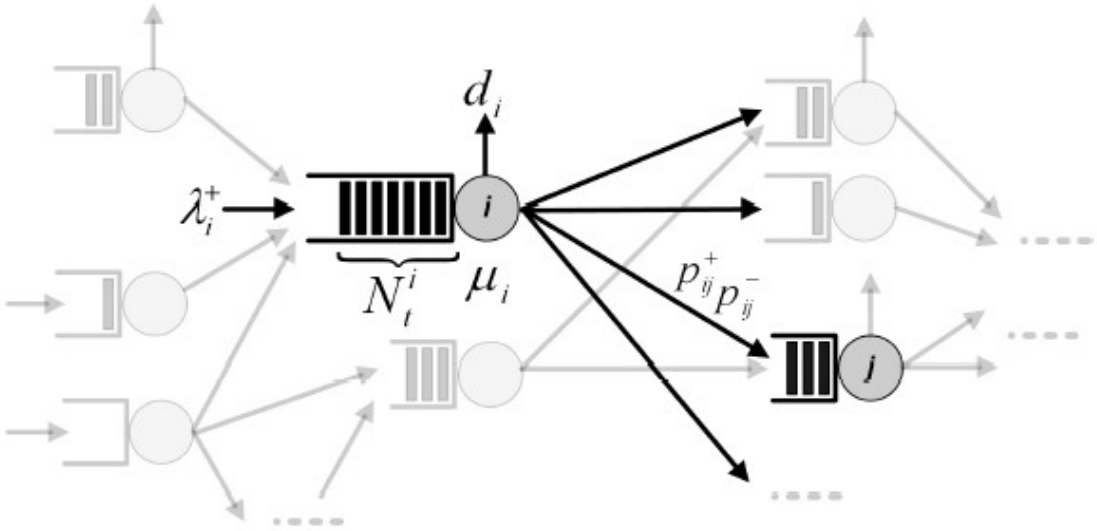


Fig. 6: Interacción de una neurona con sus vecinas

Por lo general se toma como salida las probabilidades de excitación de un subconjunto de neuronas; en nuestro caso de interés sólo será de una neurona, y su salida representará la calidad del video que se pretende evaluar.

RNN de 3 capas

La topología de una red neuronal aleatoria puede ser muy variable. Una red neuronal aleatoria de 3 capas en particular presenta un grupo de neuronas de entrada, otro grupo perteneciente a la capa oculta y uno último que es la capa de salida.

Para el problema particular tanto el número de neuronas de entrada como de salida ya están predeterminados: tantas neuronas en la capa de entrada como parámetros de entrada, es decir 2, y una neurona de salida que nos brindará "su opinión del video".

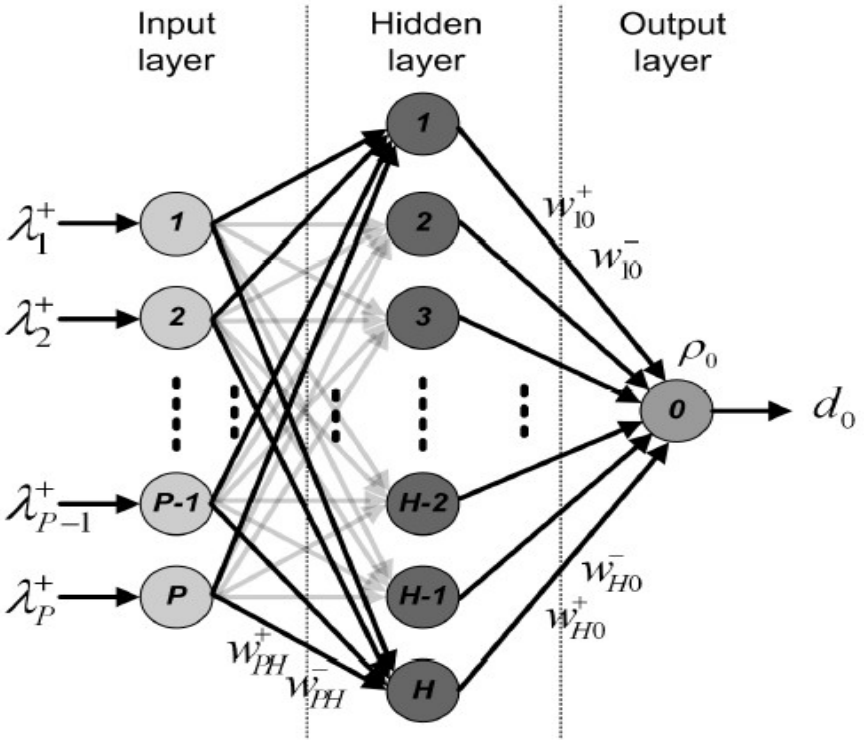


Fig. 7: RNN de 3 capas con P neuronas en la entrada, H ocultas y 1 a la salida.

Las neuronas de la capa de entrada y oculta no liberan señales a la naturaleza. Las de entrada sólo se comunican directamente con las ocultas, y estas últimas con la de salida, como se puede apreciar en la figura 7.

El problema de diseño consiste entonces en elegir los pesos $w_{PH}^+, w_{PH}^-, w_{HO}^+$ y w_{HO}^- de forma tal que la probabilidad de excitación de la neurona de salida sea lo más próxima al valor normalizado de MOS para la respectiva secuencia de video.

Más concretamente, el problema es el siguiente:

$$\arg\min MSE(\vec{w}) = \arg\min \frac{1}{100}\sum_{1}^{100}(\rho_0(\vec{\lambda_i^+}, \vec{w}) - m_i)^2 ,$$

por lo que debemos elegir los pesos que minimicen el error medio cuadrático con respecto a los valores medios de opinión normalizados.

A partir de la red de tres capas es posible obtener una expresión para las probabilidades de excitación en función de las tasas de entrada. Para las neuronas de entrada se tiene:

$$\rho_i = \frac{\lambda_i^+}{\mu_i},\ \forall i = 1..P$$

Obsérvese que es posible expresar la tasa de servicio de cada una de estas neuronas de entrada a partir de pesos:

$$\sum_{h\in H}\left(w_{ih}^+ + w_{ih}^-\right) = \mu_i \sum_{h\in H}\left(p_{ih}^+ + p_{ih}^-\right) = \mu_i,\ \forall i = 1...P$$

Análogamente, para las neuronas de la capa oculta $h \in H$ tenemos que:

$$\rho_h = \frac{\sum_i w_{ih}^+ \rho_i}{\mu_h + \sum_i w_{ih}^- \rho_i}, \forall h \in H$$

Por último, la probabilidad de excitación de la neurona de salida, que es la medida de calidad, se expresa así:

$$\rho_0 = \frac{\sum_i w_{h0}^+ \rho_h}{\mu_0 + \sum_i w_{h0}^- \rho_h}$$

Obsérvese que a entradas nulas, la función de salida es siempre 0, independientemente de los pesos de diseño. Esto no es lo buscado en nuestro caso, puesto que con pérdidas nulas la calidad es la óptima. Agregamos entonces una entrada de valor constante igual a 1. Esto es equivalente a considerar 3 neuronas a la entrada, donde una de ellas siempre recibe la entrada de 1.

Nuestro procedimiento de búsqueda de los pesos óptimos se basa en una técnica de descenso de gradiente.

Mínimo y vector gradiente

Procedemos ahora calculando el vector que contiene las derivadas parciales de la función MSE respecto a todos los pesos de la red (tanto inhibitorios como excitatorios).

Vamos a utilizar la notación I, H y O para referirnos a los conjuntos de neuronas de entrada, ocultas y de salida, respectivamente. Hay entonces 4 clases de pesos: aquellos que vinculan la capa de entrada con la

oculta, otros que vinculan esta última con la salida y en cada uno de los casos la posibilidad de que sean inhibitorios o bien excitatorios.

En general, para cada peso se tiene que:

$$\frac{\partial MSE(w)}{\partial w} = \frac{\partial}{\partial w}\frac{1}{100}\sum_{1}^{100}(\rho_0(\vec{\lambda_i^+},\vec{w}) - m_i)^2 = \frac{1}{50}\sum_{1}^{100}(\rho_0(\vec{\lambda_i^+},\vec{w}) - m_i)\frac{\partial \rho_0(\lambda_i^+, w)}{\partial w} \quad (1)$$

por lo que debemos calcular las derivadas de la probabilidad de excitación de la neurona de salida para cada una de las 4 clases de pesos de la red.

Derivada respecto de pesos w_{IH}^+

$$\frac{\partial \rho_0}{\partial w_{IH}^+} = [w_{H0}^+ \frac{\partial \rho_h}{\partial w_{IH}^+}(\mu_0 + \sum_{h\in H}\rho_h w_{H0}^-) - (\sum_{h\in H}\rho_h w_{H0}^+)\cdot w_{H0}^- \frac{\partial \rho_h}{\partial w_{IH}^+}]/(\mu_0 + \sum_{h\in H}\rho_h w_{H0}^-)^2, \ y$$

$$\frac{\partial \rho_h}{\partial w_{IH}^+} = (\rho_i - \frac{\rho_i}{\mu_i}w_{IH}^+)(\mu_h + \sum_{i\in I}\rho_i w_{H0}^-) + \frac{\rho_i}{\mu_i}w_{IH}^- \sum_{i\in I}\rho_i w_{IH}^+$$

Derivada respecto de pesos w_{IH}^-

$$\frac{\partial \rho_0}{\partial w_{IH}^-} = [w_{HO}^+ \frac{\partial \rho_h}{\partial w_{IH}^-}(\mu_0 + \sum_{h\in H}\rho_h w_{H0}^-) - w_{H0}^- \frac{\partial \rho_h}{\partial w_{IH}^-}\sum_{h\in H}\rho_h w_{H0}^+]/(\mu_0 + \sum_{h\in H}\rho_h w_{H0}^-)^2, \ y$$

$$\frac{\partial \rho_h}{\partial w_{IH}^-} = -[(\rho_i - \frac{\rho_i}{\mu_i}w_{IH}^-)\sum_{i\in I}\rho_i w_{IH}^+) + \frac{\rho_i}{\mu_i}w_{IH}^+(\mu_h + \sum_{i\in I}\rho_i w_{IH}^-)]/(\mu_h + \sum_{i\in I}\rho_i w_{IH}^-)^2$$

Derivada respecto de pesos w_{HO}^+

$$\frac{\partial \rho_0}{\partial w_{HO}^-} = [(w_{HO}^+ \frac{\partial \rho_h}{\partial w_{HO}^+} + \rho_h)(\mu_0 + \sum_{h\in H}\rho_h w_{H0}^-) - w_{H0}^- \frac{\partial \rho_h}{\partial w_{HO}^+}\sum_{h\in H}\rho_h w_{H0}^+]/(\mu_0 + \sum_{h\in H}\rho_h w_{H0}^-)^2, \ y$$

$$\frac{\partial \rho_h}{\partial w_{HO}^+} = -\sum_{i\in I}\rho_i w_{IH}^+ /(\mu_h + \sum_{i\in I}\rho_i w_{IH}^-)^2$$

Derivada respecto de pesos w_{HO}^+

$$\frac{\partial \rho_0}{\partial w_{IH}^-} = [w_{HO}^+ \frac{\partial \rho_h}{\partial w_{HO}^-}(\mu_0 + \sum_{h\in H}\rho_h w_{H0}^-) - (\rho_h + w_{H0}^- \frac{\partial \rho_h}{\partial w_{HO}^-})\sum_{h\in H}\rho_h w_{H0}^+]/(\mu_0 + \sum_{h\in H}\rho_h w_{H0}^-)^2, \ y$$

$$\frac{\partial \rho_h}{\partial w_{HO}^-} = \frac{\partial \rho_h}{\partial w_{HO}^+}$$

Sustituyendo cada una de estas derivadas en (1) es posible plantear analíticamente la expresión de las derivadas de la función MSE con respecto a cada uno de los pesos (que son los parámetros de diseño de la red).

Es importante aclarar que el objetivo posterior es una búsqueda de los pesos de diseño óptimos. Puesto que la función MSE presenta un mínimo absoluto cuando las salidas de nuestra red coinciden con la función

de calidad, en los pesos óptimos se debe anular el vector gradiente de la función MSE.

El código de Matlab derivadas.m calcula las anteriores derivadas dados los pesos de diseño, retornando el vector gradiente de la función MSE.

El proceso de búsqueda se basa en una técnica simple de descenso de gradiente, que consiste en reducir los pesos en proporción con la derivada parcial de la función de error respecto de este mismo peso:

$$w_{n+1} = w_n - \eta \frac{\partial f}{\partial w} \quad \forall w$$

La constante de proporcionalidad se denomina usualmente "velocidad de aprendizaje", dado que mide de qué forma nos acercamos al cero en cada derivada parcial (un exceso de este parámetro lleva a cambios de signo de los pesos e inestabilidad).

El programa backpropagation.m realiza la técnica de descenso de gradiente, recién comentada.

Resultados

Introducción

En esta sección detallamos los resultados obtenidos de extrapolar la función de calidad tanto con RNN como con ANN.

En primera instancia detallamos el proceso de entrenamiento que fue necesario para la obtención de la salida para RNN.

Luego mostramos gráficas de la función de calidad obtenida mediante el uso de RNN, y qué tan bien se ajusta a la función de calidad subjetiva o MOS. A continuación realizamos un análisis comparativo de la técnica RNN con la clásica de redes neuronales artificiales, o ANN, aplicadas en particular al ajuste de la función de calidad para las 100 muestras de video utilizadas en la encuesta utilizada.

Proceso de entrenamiento

El entrenamiento de la RNN es una etapa central a la hora de buscar un ajuste satisfactorio entre la función de calidad y la salida de la red neuronal.

Los programas Matlab utilizados para tales efectos son ro.m, feedforward.m, backpropagation.m, Derivadas.m, Busqueda.m, Graficar.m y PSQARNN.m; todos se encuentran en el apéndice de programas.

La esencia del entrenamiento se halla en backpropagation. Aquí se aplica la técnica de descenso de gradiente ingresados la velocidad de aprendizaje, el número de neuronas ocultas y los patrones etiquetados. Esta función calcula en primera instancia las salidas para cada patrón en la entrada, llamando a la función feedforward (que para cálculos internos llama a su vez a ro.m, que es la expresión clásica de avance de una capa en RNN).

Luego, en su ciclo principal, obtiene al vector gradiente a partir de Derivadas.m, y actualiza los pesos mediante un descenso de gradiente con el parámetro de aprendizaje ingresado, tal como se mencionó previamente.

Este ciclo termina cuando se logra una norma pequeña del vector gradiente, o bien cuando se alcanza un número limitado de iteraciones (para evitar loop infinito).

La función backpropagation retorna así el mse que se obtiene a partir de los mejores pesos de diseño, y estos últimos.

Una particularidad adicional de esta función principal es que cada salida toma varios minutos en obtenerse, por lo que la búsqueda tanto del número de neuronas ocultas como del eta óptimo no son triviales.

Para realizar una búsqueda más automática de los parámetros de diseño, implementamos el programa Busqueda.m, que sencillamente recibe dos rangos para eta y el número de neuronas, y discretiza éstos para llamar a la función backpropagation en el ciclo principal. La búsqueda se basa entonces en retener los pesos que logran el mejor mse hasta el momento.

Naturalmente, si cada corrida de backpropagation toma unos 3 minutos, discretizando los 2 intervalos en diez ranuras esperaríamos unas 5 horas, tiempo excesivo para el proceso de entrenamiento...

Optamos en consecuencia en una búsqueda inicial del mejor número de neuronas mediante tanteos intermedios, fijando valores de eta, incrementando el número de neuronas y observando el mse a la salida (dado que se seleccionan pesos iniciales al azar en backpropagation, el mse de cada salida es distinto ante entradas iguales, por lo que debíamos aplicar la misma entrada más de una vez y promediar).

En este proceso, el número que aparentaba mejor mse (y por tanto mejor estimación de la función de calidad) es de 7 neuronas ocultas. De ahí en más buscamos el parámetro de aprendizaje para este número de neuronas ocultas. A continuación hicimos tanteos simples para estudiar básicamente en qué rangos se hallan los mejores valores de eta; de aquí deducimos que parámetros mayores que 1 llevan frecuentemente a inestabilidad y cambios bruscos de la magnitud del gradiente.

Para una búsqueda más detallada del parámetro de aprendizaje, hemos hecho uso de 19 PCs corriendo en paralelo la función Busqueda.m, donde en cada corrida se corta el intervalo ingresado en 10 partes, y para más detalles la salida brindaba el vector de 11 valores del mse.

Para las PCs 1 a 10, se ingresaron los intervalos $[i/100,(i+1)/100],\ i=0..9$ (más precisamente, la PC 1 tiene como entradas el rango $[1/1000,1/100]$, pues no lleva a descender el vector gradiente el caso de aprendizaje nulo), y en las PCs 11 a la 19 los intervalos respectivos $[i/10,(i+1)/10)],\ i=1..9$.

Este proceso de búsqueda tomó poco más de una hora. La PC número 1 fue la afortunada, por retornar el menor mse, que valió 0.0458.

Sin embargo, hemos realizado un programa de búsqueda adicional para realizar tanteos, ya teniendo presente que a valores de aprendizaje pequeños se puede obtener buenos resultados. Luego de diversas pruebas, para eta de 0.001 logramos un mse de 0.0244. Más detalles numéricos y gráficos se hallan a continuación.

Resultados del entrenamiento

RNN:
Datos de entrada:

- $\eta = 0{,}001$
- Neuronas ocultas (1 capa oculta) = 7

Datos de salida:

- MSE = 0,0244
- Pesos finales (la última fila de pesos es para bias, en el caso de la entrada):
  - WIH+ =

0.55414512489306 -5.00009988844693 8.85640631935269 0.18565242564168
-0.28057580190413 -0.56561347275853 -0.12905189372623 0.44807469675672
-0.43338863994715 0.25805355335081 -0.39520642937622 -0.34627995627305

-0.51811417349503 0.47889334300038 0.25809351627940
0.41672231061180 0.14341426585635 0.08178519985748
0.05981478176555 -0.24470181367041 0.09406454711820

  - WIH- =

0.17321179117774 -0.64602182078295 -10.88722705302268 -0.53531331261944
-0.47353313906258 1.28676257793710 -2.52959499563690 0.07687305958748
0.00895649014793 -5.22077024589953 5.04539787128715 0.37442149235352

-0.16547790438574 0.24324553917888 0.25049955993492
0.56462711204041 0.59961995726050 -0.43961683871975
0.21147106167951 0.72998349361978 -0.83306037628772

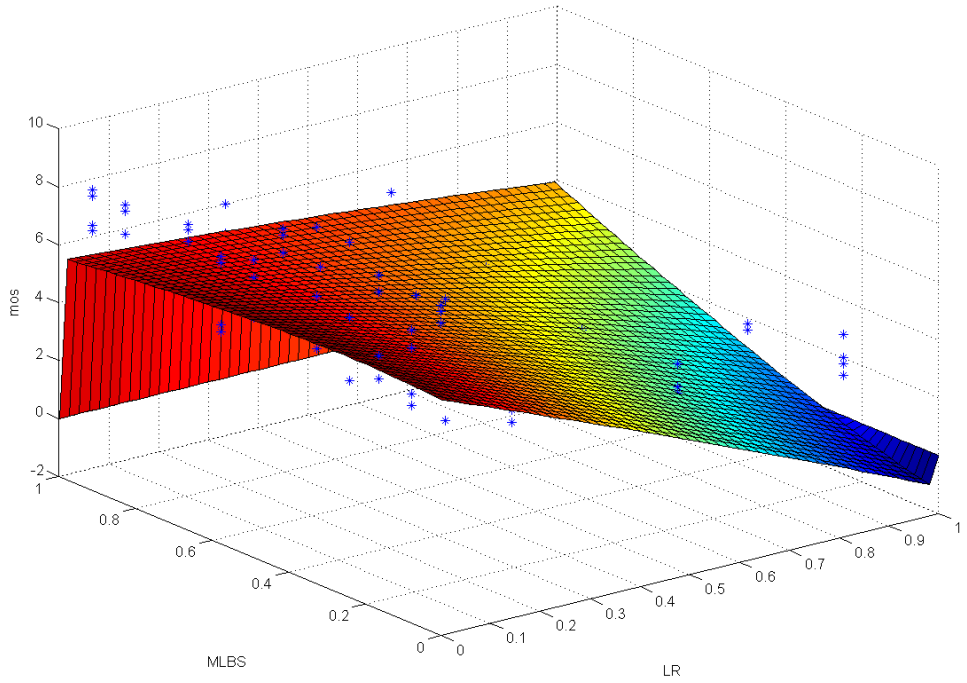
  - WHO+ =

-0.22874077538065 1.42175886736763 -17.08239198923798 -0.11533929592836
-0.02421767986251 -0.25303773564205 0.05293666172057

  - WHO- =

-0.42396635998212 9.79599740017838 -11.65669087005136 -0.56664235354607
-0.27303198631376 -0.39990250136826 -0.74971480498465

Gráfica obtenida:



ANN

Datos de entrada:

- $\eta = 0{,}02$
- Neuronas ocultas (1 capa oculta) = 7

Datos de salida:

- MSE = 0.0621
- Pesos finales:
  - WIH =

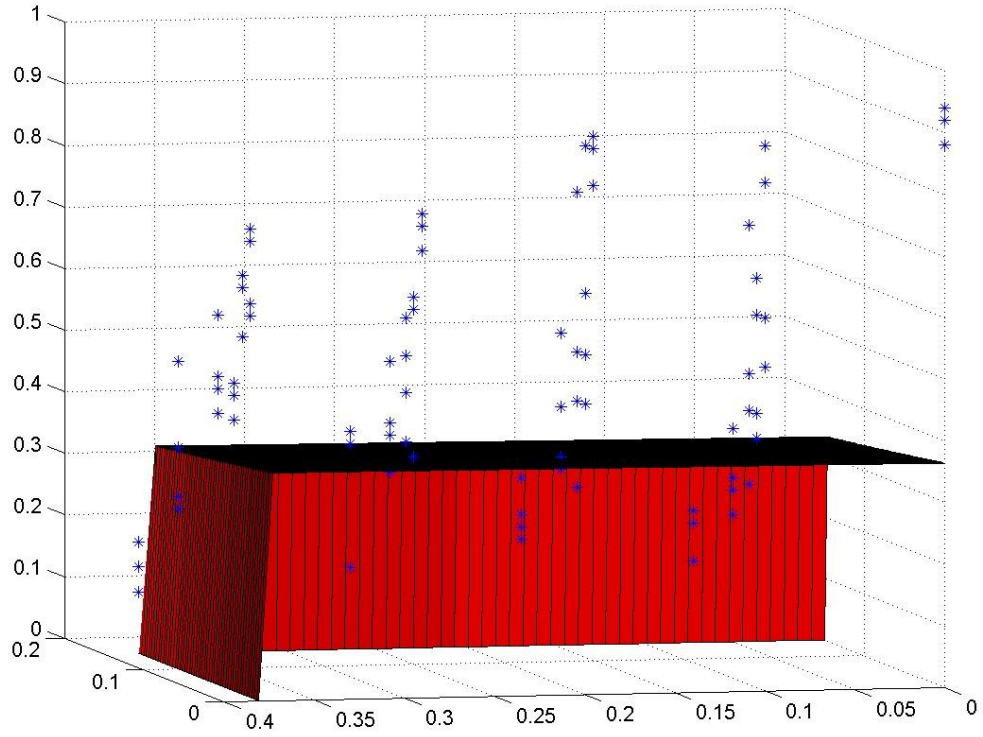
0.0113	0.1864	0.5091
-0.0058	0.2019	-0.5942
1.9466	1.6706	0.1193
-0.2798	-0.7131	-0.1330
0.0299	0.2532	0.1607
0.0031	-0.0107	0.1852

-0.0354 0.2481 -0.0067

■ WHO =

1.0109 0.0436 -0.0532 -0.4553 -0.0233 -0.0618 -0.0189 -0.0467

Gráfica obtenida:



Conclusiones

El proceso de diseño mostró ciertas complejidades, fundamentalmente debidas a los tiempos computacionales. Sin embargo, se puede apreciar que a efectos prácticos, la superficie de estimación a la calidad obtenida mediante RNN es satisfactoria.

Trabajos previos muestran que RNN presenta mejor performance que ANN[2][7]. En nuestro caso no es posible dar tal afirmación, pues no le hemos dado igual énfasis en la búsqueda de parámetros óptimos para ANN, habiéndonos centrado en el entenamiento de la RNN. Sin embargo, las superficies encontradas resultan claramente mejor en RNN.

Es interesante destacar que la tesis de doctorado en la que nos hemos basado [1] considera incluso aceptable una red de 2 capas (es decir, ninguna capa oculta), y los pesos de la red se hallan mediante los métodos estándar de resolución de mínimos cuadrados lineales.

Esto último muestra que la precisión de la aproximación es muy variable según la aplicación, y en algunos casos se puede valorar la simplicidad.

Un problema abierto

Para finalizar dejemos claro en qué contexto se aplica hoy en día la medida pseudosubjetiva de calidad estudiada en esta monografía, y presentaremos un problema abierto referente a PSQA y redes P2P.

Las redes de pares han crecido brutalmente en estos últimos tiempos. Este crecimiento se justifica básicamente por la necesidad de compartir información entre los usuarios (video, audio, textos, transacciones y otros), como también por el incremento del ancho de banda a nivel mundial, que se explica a partir de la política promotora de tarifas planas por parte de los proveedores de servicios de internet.

Por su parte, las redes de pares presentan un dinamismo único: los usuarios se conectan y desconectan a su deseo y en forma asíncrona, como también descargan cierto número de contenidos que prefieren[8]. Estas libertades son necesarias del punto de vista de los usuarios, aunque también generan interesantes dificultades a la hora de gestionar la red. Es suficiente mencionar que entre un 50 y un 90% del flujo existente en los enlaces internacionales es de aplicaciones P2P, hecho que atrae no sólo del punto de vista académico sino también económico, puesto que los proveedores del servicio pagan por el malgasto de estos enlaces.

Diversas estrategias se consideran para lograr que el flujo P2P sea una solución para usuarios y a la vez no sea un problema para ISPs. Entre ellos se destacan packet-shaper[9], cacheado de distintas aplicaciones P2P, localizaciones de flujos con LiteLoad[10] y participación proactiva del proveedor (P4P, más detalles en la referencia [11]). Las dos primeras son fuertemente invasivas, en el sentido que ciertos usuarios serán perjudicados pues sus flujos no serán considerados prioridad (serán cacheados o bien llevados al último lugar de una cola con prioridades). LiteLoad tiene la ventaja de aliviar los enlaces internacionales mediante la presión a los usuarios de comunicarse entre vecinos de su mismo ISP, aunque a la idea inicial le falta cierta madurez.

Una alternativa interesante y más madura es P4P, que permite centralizar la información en un servidor llamado ITracker. Este último

logra determinar la topología de la red a partir de consultas periódicas de los pares que se hallan en la red.

Aquí viene el problema. Pensemos por un momento que cierto usuario se conecta y solicita un contenido, ¿con cuántos pares lo comunicamos y con quiénes (los "mejores" pares o los peores). Por otro lado, ¿cómo realizamos el compromiso de eficiencia de ancho de banda y la calidad percibida por los usuarios? Sí existe una gran variedad de trabajos que reducen el diseño y toma de decisiones en la comunicación a un problema de optmización[8][11], de forma muy concreta.

Sin embargo, aún no hay trabajos que partiendo de la topología de la red procuren maximizar la calidad pseudosubjetiva PSQA esperada entre usuarios, sujeta a las condiciones de factibilidad con el ancho de banda disponible.

Conclusiones Finales

La técnica de evaluación pseudosubjetiva denominada PSQA se muestra muy valiosa una vez que la medida en tiempo real es requisito, desplazando las clásicas técnicas objetivas y subjetivas. El tiempo que toma la evaluación es el de evaluar una simple función racional de las entradas, por ello se puede hablar de tiempo real. Hemos visto en este trabajo su aplicación para la evaluación de calidad de video, con previa encuesta realizada por expertos.

Los resultados fueron muy satisfactorios, si comparamos la función de calidad real con aquella obtenida mediante la "opinión" de la neurona de salida en RNN. Hemos obtenido un error mse del orden de 2 centésimos, lo cual es similar a lo obtenido en [1].

Finalmente, vale observar que la medida de calidad PSQA no está ligada a video, sino que también se puede utilizar para otras aplicaciones, como lo son para audio y voz (en particular VoIP). La técnica de evaluación de calidad se muestra válida no sólo para fines académicos, sino también como una interesante medida de la calidad percibida por los usuarios, y muy prometedora para ser usada en masividad para el diseño de redes, en un futuro no lejano.

BIBLIOGRAFÍA

[1] Quality-centric design of Peer-to-Peer systems for live-video broadcasting, Tesis de doctorado de Pablo Rodríguez Bocca

[2] A Study of Real-Time Packet Video Quality Using Random Neural Networks - Samir Mohamed, Gerardo Rubino; IEEE Transactions on Circuits and Systems for Video Technology, Vol.12, No12, Diciembre 2002.

[3] Evaluating Users' Satisfaction in Packet Networks Using Random Neural Networks - Gerardo Rubino, Pierre Tirilly, Martín Varela; INRIA/IRISA, Rennes

[4] Quality-centric design of Peer-to-Peer systems for live video broadcasting – Pablo Rodríguez-Bocca. Tesis de Doctorado. Université de Rennes

[5] Pattern Classification - Richard O. Duda, Peter E. Hart, David G. Stork; Wiley-Interscience

[6] A tutorial on Random Neural Networks – G. Rubino Irisa/Inria, Rennes, France LANC'05, Cali, Colombia

[7] S. Mohamed, G. Rubino, H. Afifi, and F. Cervantes, Real-time video quality assessment in packet networks: A neural network model, Las Vegas, NV, June 2001.

[8] Redes de Contenido: Taxonomía y Modelos de evaluación y diseño de los mecanismos de descubrimiento de contenido. Tesis de maestría de Pablo Rodríguez Bocca

[9] Sitio web de PacketShaper. http://www.bluecoat.com/products/packetshaper

[10] LiteLoad: Content Unaware Routing for Localizing P2P protocols. Shay Horovitz y Danny Dolev.

[11] P4P: Proactive Provider Participation for P2P. Haiyong Xie, Arvind Krishnanmurthy, Yang Richard, Yang Avi Silberschatz

Programas en Matlab

Introducción

Todos los programas fueron realizados en Matlab versión 7. Este apartado se ordena de la siguiente manera. En primera instancia brindamos la cabecera de cada uno de los scripts, explicando en pocas palabras el porqué de su existencia y en qué punto del procedimiento PSQA tiene lugar (según la numeración del 1 al 7 del capítulo PSQA). Finalmente, adjuntamos todos los códigos. Vale agregar que cada script tiene comentarios al inicio para mayor facilidad de su comprensión y funcionamiento.

Cabecera de los scripts

function b = ber(p)
Elemental. Simula una distribución de Bernouli de parámetro p.

function e = gilbert(LR,MLBS,NCuadros)
Emula el proceso de pérdidas de cuadros de video según el modelo de Gilbert. Es útil para distorsionar secuencias de video controladamente (elemento del punto 2 en el proceso PSQA).

function r = rand100(a,b,c,d)
Elemental.Genera 100 pares aleatorios a la salida, donde la primera coordenada tiene como extremos a y b, y la segunda c y d
(punto 3 del proceso PSQA, para distorsionar 100 videos originales).

function b = beta2(q)
Es un test de normalidad. Se ingresa el vector de muestras q y retorna 0 si rechaza, o bien 1 si no rechaza (punto 4 del proceso PSQA: filtrado).

function f = filtro(opinion)
Recibe una matriz con las opiniones de cada video del panel, y retorna un vector que tiene un 0 en el lugar i-ésimo si el miembro i del panel se apartó significativamente de la opinión media, y sus muestras se deben eliminar; 1 en caso contrario (útil para el punto 4 del proceso PSQA, de filtrado).

De aquí en más, todos los scripts se utilizan para el diseño y entrenamiento de la RNN.

function r=ro(lambda,wMas,wMenos,mu,roAnterior)
Tan solo efectúa una combinación racional en las entradas, muy frecuente en redes neuronales (expresión de "ro" según tasas de entrada y salida).

function [roentrada,roocultas,rosalida,mu,muH] = feedforward (WIHMenos,WIHMas,WHOMenos,WHOMas,entrada)
Dada una cierta entrada y los pesos de una red neuronal aleatoria, retorna las probabilidades de excitación y tasas de servicio.

function salida = derivadas
(WIHMenos, WIHMas, WHOMenos, WHOMas,entrada,mos)
Retorna el vector gradiente de la función MSE evaluado en los pesos de entrada.

function [wIHMenos,wIHMas,wHOMenos,wHOMas,mse] = backpropagation(etiquetas,patrones,Nocultas,eta)
Realiza el entrenamiento propiamente dicho de la red mediante un descenso de gradiente. La variable de entrada eta es la velocidad de aprendizaje. Las salidas son los pesos que logran un bajo mse, y este último valor.

function [eta, Nocultas] = Busqueda(etiquetas,patrones,ocultas_inf,ocultas_sup,eta_inf,eta_sup)

Realiza una búsqueda de la mejor velocidad de entrenamiento y el número óptimo de neuronas ocultas dentro de un rango de búsqueda especificado.

function [entrenamiento, validacion, prueba] = separar(patrones, porc_entrenamiento, porc_validacion)
Separa los patrones en tres subconjuntos tomados al azar, según los porcentajes ingresados en las entradas.

Códigos en Matlab

```
function b = ber(p)
% b = ber(p) simula la distribución Bernouli de parámetro p
% Vale 1 con probabilidad p

 b = rand < p;
```

```
function e = gilbert(LR,MLBS,NCuadros)
% e = gilbert(LR,MLBS,NCuadros) emula las pérdidas del video
% Según el proceso de Gilbert (Markov de 2 estados)
% Retorna en e la evolución de los cuadros del video,
% donde 0 es "cuadro dañado" y 1 es "cuadro transmitido"

q = 1/MLBS;
p = 1/MLBS * (LR/(1-LR));
e = [ber(q/(p+q))];

for i=2:NCuadros
   if e(i-1)==1
      e(i) = ber(1-p);
   else
      e(i) = ber(q);
   end
end
```

```
function r = rand100(a,b,c,d)
% r = rand100(a,b,c,d)
% Genera un vector aleatorio de 100 muestras
% a y b son los extremos del intervalo de la 1ª coordenada
% c y d son los extremos del intervalo de la 2º coordenada

r     =      rand(100,2);
r(:,1) = a + (b-a)* r(:,1);
r(:,2) = c + (d-c)* r(:,2);
```

```
function b = beta2(q)
% b = beta2(q) es un test de normalidad
% la salida vale 0 si se rechaza la hipótesis de
% normalidad, 1 si no.
% Se debe ingresar el vector q de muestras

n  = length(q);
qm = mean(q);
m2 = mean((q-qm).^2);
if (m2==0)
   b=1;
else
   m4 = mean((q-qm).^4);
   x  = m4/m2^2;
   b  = (x >= 2)&&(x <= 4);
end
```

```
function f = filtro(opinion)
% f = filtro(opinion)
% Reliza un filtrado de las muestras "malas".
% --> opinion(i,j) tiene la opinion del miembro j
% sobre el video i.
% --> f(j)=1 si el miembro j opina "bien", 0 si no.

%%% Inicialización %%%
Nvideos    = length(opinion(:,1));
Npersonas  = length(opinion(1,:));

P = zeros(Npersonas,1);
Q = zeros(Npersonas,1);

opinion = opinion./10; % normalización

for i=1:Nvideos
   mos(i)   = mean(opinion(i,:));
   delta(i) = sqrt(sum((opinion(i,:)-mos(i)).^2)/(Npersonas-1))
end
```

```
%%% Ciclo principal  %%%
% Contamos número de veces que se aparta cada miembro del panel,
% tanto por arriba (P(i)), como por abajo (Q(i))
for j=1:Npersonas
   contador = 0;
   for i=1:Nvideos
      if beta2(opinion(i,:))==1
         if (opinion(i,j)>= mos(i)+ 2*delta(i))
            P(j) = P(j) + 1
         end
         if (opinion(i,j)<= mos(i)- 2*delta(i))
            Q(j) = Q(j) + 1
         end
      else
         if (opinion(i,j)>= mos(i)+ sqrt(20)*delta(i))
            P(j) = P(j) + 1
         end
         if (opinion(i,j)<= mos(i)- sqrt(20)*delta(i))
            Q(j) = Q(j) + 1
         end
      end
   end
end

% Si el miembro i del panel tuvo muchos apartamientos y además
% con sesgo (videos sobrecalificados o sub-calificados),
% se debe eliminar(f(i)=0)
for i=1:Npersonas
f(i) = 1-(((P(i)+Q(i))/Nvideos > 0.05) && (abs((P(i)-Q(i))/(P(i)+Q(i))) <
0.3));
end
```

```
function r=ro(lambda,wMas,wMenos,mu,roAnterior)
% Realiza una combinación racional de las entradas
% que es de muy frecuente uso en RNN

r = (lambda + roAnterior*(wMas))/(mu + roAnterior*(wMenos));
```

```
function   [roentrada,roocultas,rosalida,mu,muH] =
feedforward(WIHMenos,WIHMas,WHOMenos,WHOMas,entrada)
% Calcula las salidas de la RNN y tasas de servicio
% dados un vector de entrada
% y los pesos actuales de la red

%%% Inicialización %%%
Nocultas = length(WIHMenos(1,:));
Nentradas = length(entrada);
Nsalida = 1;
mu0 = 0.01;

% Cálculo de las tasas de salida a partir de la definición de
% los pesos, y de los pesos de entrada.
for i=1:Nentradas
   mu(i) = sum(WIHMenos(i,:)) + sum(WIHMas(i,:));
   roentrada(i) = entrada(i)/mu(i);
end

% Cálculo de tasas de servicio a la salida a partir de la
% definición de los pesos de salida, y fórmula para ro de salida
for i=1:Nocultas
   muH(i) = WHOMas(i) + WHOMenos(i);
   roocultas(i) = ro(0,WIHMas(:,i),WIHMenos(:,i),muH(i),roentrada);
end

rosalida = ro(0,WHOMas,WHOMenos,mu0,roocultas);
```

```
function salida = derivadas(WIHMenos, WIHMas, WHOMenos,
WHOMas,entrada,mos)
% Calcula las derivadas de la funcion MSE con respecto a los pesos.
% La salida tiene:
% Las derivadas respecto de los pesos input-hidden positivos en las
% primeras Nocultas*Nentradas
% Las derivadas respecto de los pesos input-hidden negativos en las
% siguientes Nocultas*Nentradas
% Las derivadas respecto de los pesos hidden-output negativos en las
% siguientes Nocultas
% Las derivadas respecto de los pesos hidden-output positivos en las
% siguientes Nocultas
% Largo total: Nocultas*(2*Nentradas+2)
```

```
%%% Inicialización de variables %%%
Nocultas = length(WIHMenos(1,:));
Nentradas = length(WIHMenos(:,1));
Nsalida = 1;
Nmuestras = length(mos);
muO = 0.25;

roentrada = zeros(Nmuestras, Nentradas);
roocultas = zeros(Nmuestras, Nocultas);
rosalida  = zeros(Nmuestras, Nsalida);
muI       = zeros(Nmuestras, Nentradas);
muH       = zeros(Nmuestras, Nocultas);
salida = [];


% Cálculo de las salidas respectivas a las entradas
% y los "ro" de cada etapa de la red:
for i=1:Nmuestras
[roentrada(i,:),roocultas(i,:),rosalida(i,:),muI(i,:),muH(i,:)] =
feedforward(WIHMenos,WIHMas,WHOMenos,WHOMas,entrada(i,:));
end

%%%% Calculo de derivadas de la funcion error %%%%

% Derivadas según pesos wih+
for i=1:Nentradas
   for h=1:Nocultas
      derivada = 0;
      for m=1: Nmuestras
der_i_h = [(roentrada(m,i)-
roentrada(m,i)*(WIHMas(i,h)')/muI(m,i))*(muH(m,h)
+sum(roentrada(m,:).*(WIHMenos(:,h)')))+
(sum(roentrada(m,:).*(WIHMas(:,h)'))*(roentrada(m,i)*(WIHMenos(i,h)')/
muI(m,i)))]/(muH(m,h)+sum(roentrada(m,:).*(WIHMenos(:,h)')))^2;
derivada = derivada + 2*(rosalida(m)-mos(m))*(1/
(muO+sum(roocultas(m,:).*WHOMenos(h,:)')))^2*...
[WHOMas(h)*der_i_h*(muO+sum(roocultas(m,:).*WHOMenos(h,:)'))-
sum(roocultas(m,:).*WHOMas(h,:)')*WHOMenos(h,:)*der_i_h];
      end
      salida = [salida derivada];
   end

end
```

```
% Derivadas según pesos wih-
for i=1:Nentradas
   for h=1:Nocultas
      derivada = 0;
      for m=1: Nmuestras
      der_i_h(m) = [-(roentrada(m,i)*(WIHMas(i,h)')/muI(m,i))*(muH(m,h)
+sum(roentrada(m,:).*(WIHMenos(:,h)')))-
sum(roentrada(m,:).*(WIHMas(:,h)'))*(roentrada(m,i)-
(roentrada(m,i)*(WIHMenos(i,h)')/muI(m,i)))]/(muH(m,h)
+sum(roentrada(m,:).*(WIHMenos(:,h)')))^2;
      derivada = derivada + 2*(rosalida(m)-mos(m))*(1/
(muO+sum(roocultas(m,:).*(WHOMenos(h,:)'))))^2*[WHOMas(h)*der_i_
h(m)*(muO+sum(roocultas(m,:).*(WHOMenos(h,:)')))-
sum(roocultas(m,:).*(WHOMas(h,:)'))*WHOMenos(h,:)*der_i_h(m)];
      end
      salida = [salida derivada];
   end
end

% Derivadas según pesos who+
for h=1:Nocultas
   derivada = 0;
   for m=1: Nmuestras
      der_i_h(m) = -sum(roentrada(m,:).*(WIHMas(:,h)'))/(muH(m,h)
+sum(roentrada(m,:).*(WIHMenos(:,h)')))^2;
      derivada = derivada + 2*(rosalida(m)-mos(m))*(1/
(muO+sum(roocultas(m,:).*(WHOMenos(h,:)'))))^2*[(der_i_h(m)*WHOM
as(h)+roocultas(m,h))*(muO+sum(roocultas(m,:).*(WHOMenos(h,:)')))-
sum(roocultas(m,:).*(WHOMas(h,:)'))*(WHOMenos(h,:)*der_i_h(m))];
   end
   salida = [salida derivada];
end

% Derivadas segun pesos who-
for h=1:Nocultas
   derivada = 0;
```

```
    for m=1:Nmuestras
        der_i_h(m) = -sum(roentrada(m,:).*(WIHMas(:,h)'))/(muH(m,h)
+sum(roentrada(m,:).*(WIHMenos(:,h)')))^2;
        derivada = derivada + 2*(rosalida(m)-mos(m))*(1/
(muO+sum(roocultas(m,:).*(WHOMenos(h,:)'))))^2*[(der_i_h(m)*WHOM
as(h))*(muO+sum(roocultas(m,:).*(WHOMenos(h,:)')))-
sum(roocultas(m,:).*(WHOMas(h,:)'))*(roocultas(m,h)
+WHOMenos(h,:)*der_i_h(m))];
    end
    salida = [salida derivada];
end
```

```
function [wIHMenos,wIHMas,wHOMenos,wHOMas,mse] =
backpropagation(etiquetas,patrones,Nocultas,eta)

% Entrena una RNN de 3 capas con dos entradas.
% "patrones" tiene un patrón por fila.
% "Nocultas" dice cuántas neuronas ocultas se quiere


%%% Inicialización %%%
tic  % contabilizamos el tiempo total del script
Npatrones = length(patrones(:,1));
Nentradas = length(patrones(1,:));

%Normalización
for i=1:Nentradas
    patrones(:,i)=patrones(:,i)/max(patrones(:,i));
    etiquetas(i) =etiquetas(i)/max(etiquetas);
end


%Inicializamos los pesos
wHOMenos = (1/(sqrt(Nocultas)))*(1-2*rand(Nocultas,1));
wHOMas = (1/(sqrt(Nocultas)))*(1-2*rand(Nocultas,1));
wIHMenos = (1/(sqrt(Nentradas)))*(1-2*rand(Nentradas,Nocultas));
wIHMas = (1/(sqrt(Nentradas)))*(1-2*rand(Nentradas,Nocultas));

tol=0.1;
cond=tol+1;
k=0;
```

```
%%% Ciclo principal %%%
%% Entrenamiento hasta tener bajo gradiente %%
while((cond>=tol) & (k<=500))
k=k+1;
Gradiente = derivadas(wIHMenos, wIHMas, wHOMenos,
wHOMas,patrones,etiquetas);

% Retornamos pesos en forma matricial
% Realizamos a continuación la conversión vector --> matriz
DerIHMas = zeros(Nentradas,Nocultas);
   for i=1:Nentradas
      for j=1: Nocultas
         DerIHMas(i,j) = Gradiente(j+(i-1)*Nocultas);
      end
   end

DerIHMenos = zeros(Nentradas,Nocultas);
   for i=1:Nentradas
      for j=1: Nocultas
  DerIHMenos(i,j)=Gradiente(Nentradas*Nocultas+j+(i-1)*Nocultas);
      end
   end

DerHOMas = zeros(Nocultas,1);
      for j=1: Nocultas
  DerHOMas(j,1) = Gradiente(2*Nentradas*Nocultas + j);
      end

DerHOMenos = zeros(Nocultas,1);
     for j=1: Nocultas
  DerHOMenos(j,1) = Gradiente(2*Nentradas*Nocultas + Nocultas + j);
     end

% Descenso de gradiente según el parámetro eta:
wHOMenos = wHOMenos - eta*DerHOMenos;
wHOMas = wHOMas - eta*DerHOMas;
wIHMenos = wIHMenos - eta*DerIHMenos;
wIHMas = wIHMas - eta*DerIHMas;

% Cálculo del mse (mean square error)
   mse = 0;
   for i=1:Npatrones
```

```
   [roentrada,roocultas,rosalida,mu,muH]=feedforward(wIHMenos,wIHMa
s,wHOMenos,wHOMas,patrones(i,:));
   mse = mse + (rosalida-etiquetas(i))^2;
   end
   mse = mse/Npatrones;
cond = norm(Gradiente,2); % Parada con gradiente pequeño
end  % Fin del ciclo principal
toc  % Fin del programa y del conteo del tiempo
```

```
function [eta, Nocultas] =
busqueda(etiquetas,patrones,ocultas_inf,ocultas_sup,eta_inf,eta_sup)
% Busca segun el rango del parametro de aprendizaje eta y
% rango de neuronas ocultas ingresados, el par de valores de esos rangos
% con el mse (mean square error) menor.

%%% Inicialización %%%
mse_optimo = 10
eta_optimo = eta_inf;
ocultas_optimo = ocultas_inf;
paso_eta = (eta_sup-eta_inf)/10;

%%% Ciclo Principal %%%
% Recorrido para buscar eta óptimo y Nocultas óptimo
% dentro del rango de entrada especificado
for eta=eta_inf:paso_eta:eta_sup
   for ocultas=ocultas_inf:ocultas_sup
      mse_nuevo = 1
[wIHMenos,wIHMas,wHOMenos,wHOMas,mse_nuevo]=backpropagation
(etiquetas,patrones,ocultas,eta);
if (mse_nuevo < mse_optimo)
   mse_optimo = mse_nuevo
   ocultas_optimo = ocultas;
   eta_optimo = eta
end
   end
end
eta = eta_optimo;
Nocultas = ocultas_optimo;
```

```
function [entrenamiento, validacion, prueba] = separar(patrones,
porc_entrenamiento, porc_validacion)
% Separa los patrones en tres subconjuntos tomados
% al azar, según los porcentajes ingresados en las entradas

%%% inicialización %%%
Npatrones = length(patrones(:,1));
permutacion = randperm(Npatrones);
Nentrenamiento = round(Npatrones*porc_entrenamiento/100);
Nvalidacion    = round(Npatrones*porc_validacion/100);
Nprueba        = Npatrones - Nentrenamiento - Nvalidacion;

%%% Subconjuntos de salida %%%
entrenamiento = patrones(permutacion(1:Nentrenamiento),:);
validacion    =
patrones(permutacion(Nentrenamiento+1:Nentrenamiento+Nvalidacion),:);
prueba= patrones
(permutacion(Nentrenamiento+Nvalidacion+1:Npatrones),:);
```

Mostramos a continuación la terna de códigos adicionales con los que hemos logrado el mejor entrenamiento de la RNN. Es otra versión de backpropagation, psqarnn y graficar, con la misma filosofía de diseño que antes, pero con más poder de visualización, hecho que nos condujo definitivamente a mejoras en el diseño.

```
% Función psqarnn
clc
clear all
close all

load -ascii patronesC.txt
load -ascii etiquetas.txt

Nocultas = 7;
eta=0.001;

[wIHMenos,wIHMas,wHOMenos,wHOMas,mse_nuevo]=backpropagation
(etiquetas,patronesC,Nocultas,eta)
pause


graficar(wIHMenos,wIHMas,wHOMenos,wHOMas,patronesC,etiquetas)
```

```
function
graficar(wIHMenos,wIHMas,wHOMenos,wHOMas,patrones,etiquetas)

Nmesh=50;
x=min(patrones(:,1)):(max(patrones(:,1))-
min(patrones(:,1)))/Nmesh:max(patrones(:,1))
y=min(patrones(:,2)):(max(patrones(:,2))-
min(patrones(:,2)))/Nmesh:max(patrones(:,2));

Z=zeros(length(x),length(y));
for i=1:Nmesh
   for j=1:Nmesh

      [roentrada,roocultas,rosalida,mu,muH]=feedforward(wIHMenos,wIH
Mas,wHOMenos,wHOMas,[x(j) y(i) 1]);

      Z(i,j)=rosalida;
   end
   %Para que muestre el porcentaje
   disp('Seeeeegunda...')
   M=Nmesh/100;
   if mod(i,M)==0
```

```
        clc
        disp(strcat(num2str(i/M),'%'))
    end
    %%%%%%%%%%%%%%Listo
end
surf(x,y,10*Z)
xlabel('LR')
ylabel('MLBS')
zlabel('mos')

hold on
plot3(patrones(:,1),patrones(:,2),10*etiquetas,'*')
```

```
function
[wIHMenos,wIHMas,wHOMenos,wHOMas,mse]=backpropagation(etiquet
as,patrones,Nocultas,eta0)

% Implementa backpropagation para un problema con 2 clases y una NN
%de 3 capas.
% "etiquetas" debera ser 1 o -1, dependiendo de si el patran esta en la
% clase w1 o w2.
% "patrones" tiene un patron por fila.
% "Nocultas" dice cuantas neuronas ocultas se quiere
tic


%Parametro de inercia
alfa=0.3;

Npatrones = length(patrones(:,1));
Nentradas = length(patrones(1,:));

%Normalización
for i=1:Nentradas
    patrones(:,i)=patrones(:,i)/max(patrones(:,i));
    etiquetas(i) =etiquetas(i)/max(etiquetas);
end


%Inicializo los pesos
wHOMenos = (1/(sqrt(Nocultas)))*(1-2*rand(Nocultas,1));
wHOMas = (1/(sqrt(Nocultas)))*(1-2*rand(Nocultas,1));
```

```
wIHMenos = (1/(sqrt(Nentradas)))*(1-2*rand(Nentradas,Nocultas));
wIHMas = (1/(sqrt(Nentradas)))*(1-2*rand(Nentradas,Nocultas));

tol=0.005;
MaxIter=700;
cond=tol+1;
mse=1e20;

   deltawHOMenosViejo = 0;
   deltawHOMasViejo = 0;
   deltawIHMenosViejo = 0;
   deltawIHMasViejo = 0;
   eta=eta0;


k=1;
while(mse>=tol && k<=MaxIter)

   Gradiente = Derivadas(wIHMenos, wIHMas, wHOMenos,
wHOMas,patrones,etiquetas);

   DerIHMas = zeros(Nentradas,Nocultas);
   for i=1:Nentradas
      for j=1: Nocultas
         DerIHMas(i,j) = Gradiente(j+(i-1)*Nocultas);
      end
   end

   DerIHMenos = zeros(Nentradas,Nocultas);
   for i=1:Nentradas
      for j=1: Nocultas
         DerIHMenos(i,j) = Gradiente(Nentradas*Nocultas + j+(i-
1)*Nocultas);
      end
   end

   DerHOMas = zeros(Nocultas,1);
      for j=1: Nocultas
         DerHOMas(j,1) = Gradiente(2*Nentradas*Nocultas + j);
      end

   DerHOMenos = zeros(Nocultas,1);
     for j=1: Nocultas
```

```
      DerHOMenos(j,1) = Gradiente(2*Nentradas*Nocultas + Nocultas + j);
      end


wHOMenosViejo = wHOMenos;
wHOMasViejo = wHOMas;
wIHMenosViejo = wIHMenos;
wIHMasViejo = wIHMas;

wHOMenos = wHOMenos -eta*DerHOMenos;
wHOMas = wHOMas -eta*DerHOMas;
wIHMenos = wIHMenos -eta*DerIHMenos;
wIHMas = wIHMas -eta*DerIHMas;


  cond = norm(Gradiente,2); %max(abs(gradJ));

   %Para que muestre el porcentaje
   M=MaxIter/100;
   if mod(k,M)==0
      clc
      disp(strcat(num2str(k/M),'% --',num2str(mse),' <->eta=',num2str(eta)))
   end
   %%%%%%%%%%%%%%Listo

   mseViejo=mse;
   k=k+1;
   mse = 0;
   for i=1:Npatrones
   [roentrada,roocultas,rosalida,mu,muH]=feedforward(wIHMenos,wIHMa
s,wHOMenos,wHOMas,patrones(i,:));
   mse = mse + (rosalida-etiquetas(i))^2;
   end
   mse = mse/Npatrones;

   if((mseViejo<0.04 ) || (cond*eta<1e-4))
      graficar(wIHMenosViejo,wIHMasViejo,wHOMenosViejo,wHOMasV
iejo,patrones,etiquetas)
      mseViejo
      mse
      wHOMenos
      wHOMas
      wIHMenos
```

```
        wIHMas
        s=menu('Seguir?','si','no');
        if(s==1)
           disp('Vos te lo perdes')
        else
           wHOMenos = wHOMenosViejo;
           wHOMas = wHOMasViejo;
           wIHMenos = wIHMenosViejo;
           wIHMas = wIHMasViejo;
           return
        end
     end
end
mse
cond
toc
```