En este capítulo se describirá brevemente que es el CSLU Toolkit, así como los elementos que lo conforman. Se hará una descripción sobre la metodología de desarrollo de reconocedores. El primer enfoque está orientado hacia redes neuronales y el segundo a HMM. En ambos se describe a detalle los elementos que componen a cada uno de ellos, así como scripts en tcl y algunos otros archivos de configuración que utilizan una sintaxis predefinida por dichos ambientes.

3.1 Antecedentes

Desde los 90's, el CSLU (*Center for Spoken Language Understanding*) ha estado trabajando en el desarrollo de nuevas herramientas para crear sistemas de lenguaje hablado. El resultado de tal esfuerzo es el CSLU Toolkit, un conjunto integrado de programas y documentación que representa el estado del arte en herramientas de investigación, desarrollo y aprendizaje acerca de sistemas del lenguaje hablado.

El Toolkit es un ambiente fácil de entender que integra un conjunto de tecnologías incluyendo reconocimiento de voz, síntesis de voz, y animación facial. Esto también permite el desarrollo rápido y fácil de aplicaciones de sistemas de información utilizados a través del teléfono. La arquitectura del Toolkit tiene tres componentes principales: un conjunto de librerías que contienen módulos de tecnologías de punta, además de los antes mencionados, también cuenta con un shell interactivo de programación y un ambiente de desarrollo rápido de aplicaciones (RAD).

El desarrollo de un sistema completo de lenguaje hablado es un proceso largo y requiere un gran esfuerzo en su diseño, prueba, y liberación para que sea una aplicación útil. Con la ayuda del Toolkit, un gran número de usuarios se ha incrementado y dentro de este ambiente son capaces de hacer prototipos, pruebas e investigación. Además los investigadores son provistos con un ambiente para desarrollar, probar y mostrar sus avances de dicha investigación. De esta manera el CSLU ha sido diseñado para:

- Permitir a los expertos en el dominio, diseñar sistemas de lenguaje hablado para aplicaciones reales, en otros lenguajes además del Inglés como Español, con una serie de herramientas fáciles de usar.
- Generar sistemas de lenguaje hablado automáticamente con un alto nivel en especificaciones de diseño.
- Aprender acerca de los sistemas de lenguaje hablado a través de un curso, incorporando las herramientas ya existentes.
- Investigar fácilmente la integración hombre-computadora en muchas tareas usando sistemas de lenguaje hablado.
- Mejorar la investigación sobre tecnologías de punta, e incorporar dichos avances de la investigación hacia, sistemas de trabajo para la evaluación de aplicaciones reales.

3.1.1 El núcleo del Toolkit

El núcleo del Toolkit consiste en un conjunto de módulos que implementan tecnología fundamental a todos los aspectos de reconocimiento de voz, síntesis de voz, etc. Estos módulos están escritos en lenguaje C y forman parte de una interfaz de programación de aplicaciones (API), que es independiente del hardware y del sistema operativo.

Los módulos contienen rutinas útiles para procesar la señal de voz, para el entrenamiento de redes neuronales y de HMM, todas ellas con una interfaz. Además contiene un shell interactivo en donde son utilizados scripts tk y tcl para el manejo de cadenas e interfaces.

3.1.2 Ambiente de desarrollo del CSLU-Toolkit para redes neuronales

Esta herramienta contiene varias funciones en C, tcl y tk que implementan los pasos generales que se necesitan para crear un reconocedor basado en redes neuronales, incluyendo lo siguiente:

- Especificar las categorías y subcategorías que la red reconocerá.
- Encontrar muchos ejemplos para cada una de estas categorías en los datos de voz.
- Entrenar la red para reconocer estas categorías.
- Evaluar el desempeño de la red usando un conjunto de prueba.

Los reconocedores son desarrollados usando un enfoque basado en frames con una red neuronal para estimar las probabilidades posteriores. Los pasos desarrollados durante el reconocimiento son:

- La onda es dividida en frames.
- Las características son calculadas para cada frame. Estas características se describen sobre el espectro de la voz en el frame actual y en un número pequeño de los frames que lo rodean.
- Las características en cada frame son clasificadas en categorías basadas en fonemas usando una red neuronal. Las salidas de la red neuronal son usadas como estimadores de probabilidad, para cada categoría fonética, del frame actual que contiene esa categoría.
- La matriz de probabilidades y un conjunto de modelos de pronunciación es usada por la búsqueda de Viterbi para determinar las palabras que son más probables.

3.1.2.1 Especificación de las categorías

Para determinar las categorías que la red clasificará, es necesario realizar los siguientes tres puntos:

- El diseñador del reconocedor necesita determinar las pronunciaciones para cada una de las palabras que serán reconocidas. Los modelos de pronunciación más exactos generalmente tendrán mejores niveles de reconocimiento.
- A menudo, usamos también modelos de fonemas dependientes del contexto, lo cual significa que un fonema es clasificado diferente dependiendo de los fonemas que lo

rodean. El contexto que lo rodea puede contener un grupo de fonemas o sólo un fonema.

- Finalmente, cuando construimos los modelos fonéticos dependientes del contexto, dividimos cada fonema para que sea reconocido en uno, dos, o tres partes. Cada segmento de subfonema corresponde a una categoría para ser reconocida. Si mantenemos a los fonemas en una parte, entonces son usados sin el contexto que los fonemas que lo rodean. Si dividimos en dos partes, entonces tendremos que la mitad del modelo del fonema es dependiente del fonema que le precede y el fonema del lado derecho es dependiente del siguiente fonema.

3.1.2.2 Entrenamiento y generación de los datos

Una vez que las categorías a entrenar han sido encontradas, y el número de ejemplos por categoría ha sido determinado, los datos actuales serán entrenados y guardados en un archivo de vectores, el cual contendrá para cada ejemplo a entrenar, las características que serán la entrada en la red neuronal y la categoría objetivo.

En el CSLU Toolkit se usan redes feed-forward de 3 capas. El número de nodos de entrada es el número de características espectrales, y el número de nodos de salida es el número de categorías a ser entrenadas.

Cuando usamos muchos ejemplos por categoría, es inevitable que algunas de las categorías tengan menos ejemplos que otras, haciendo dificil de aprender dichas categorías. Esta dificultad se presenta debido al factor de que hay mucho más ejemplos negativos[Λ] que ejemplos positivos para categorías esparcidas [HOSOM98]. El número de iteraciones en el entrenamiento usualmente es de 20 a 30 iteraciones para alcanzar el mejor desempeño.

[Λ] Los ejemplos negativos son ejemplos para los cuales la categoría que está siendo entrenada tiene como valor objetivo 0 y 1 para los ejemplos positivos.

El método de *forced alignment* puede ser usado para generar etiquetas para el entrenamiento. Para generar etiquetas iniciales usando dicho método, se usa un reconocedor de propósito general. También podemos usarlo para re-entrenar una red, en este caso usamos la mejor, la cual obtiene los resultados más altos.

Un método final para mejorar los resultados es usar el entrenamiento "*forward-backward*" o "*embedded*". En el entrenamiento con forward-backward, los destinos de la red neuronal no son valores binarios. Sino probabilidades posteriores. Estas probabilidades son determinadas usando el algoritmo de forward-backward, en el cual una red neuronal previamente entrenada es usada para calcular las probabilidades de observación.

3.1.2.3 Entrenamiento

Los pasos generales para el procedimiento de entrenamiento son:

- Crear un archivo llamado *corpora*, el cual contiene una lista maestra de cada corpus y la localización y formato de los archivos en el corpus.
- Crear un archivo de "información" para el entrenamiento, desarrollo, y prueba. Este archivo contendrá toda la información que es necesaria para lo antes mencionado, además de tener el nombre base del reconocedor, el número mínimo de ejemplos requeridos por cada categoría, e información dependiente del corpus.

3.1.2.4 Evaluación de la red

Una vez que hemos entrenado necesitamos determinar cual de las iteraciones ha sido la que ha tenido mejor desempeño en el conjunto de prueba. Para esto reconocemos cada pronunciación en el conjunto de datos de desarrollo usando los pesos de la red de cada iteración. Si el número de palabras en cada pronunciación no es conocida antes, entonces necesitamos evaluar el desempeño en cada iteración en términos de sustitución, inserción y borrado de errores. Si el número de palabras es conocido con anticipación, entonces sólo necesitamos medir los errores de sustitución, con el mismo método. La exactitud de la red siempre se mide como un *100%-(subs+ins+del)*, donde *subs* son los errores de sustitución, *ins* son los errores de inserción y *del* son los errores de omisión, todos ellos en porcentajes. Procedemos a escoger la mejor red con el nivel de exactitud en

la palabra. Y en caso de exactitudes iguales, seleccionamos la iteración con el mayor nivel de exactitud en la oración.

Para encontrar los datos para el entrenamiento se usan los siguientes scripts:

- ***Find_files.tcl***.- se utiliza para encontrar los archivos de entrenamiento, desarrollo y prueba, sólo debe ser ejecutado una vez para cada conjunto de archivos.
- ***Categories.tcl***.- con este script se generan las categorías que serán entrenadas. Con la ayuda de archivos como info, vocab y parts se crea un archivo llamado desc, el cual contiene una descripción del reconocedor para ser usado por otros scripts de entrenamiento y reconocimiento. También un archivo llamado olddesc es creado y contiene una descripción similar pero en un formato más viejo.
- ***Gen_catfiles.tcl***.- se usa para crear categorías alineadas en el tiempo de transcripciones de texto o de transcripciones fonéticas. Y pueden ser escritas en un archivo maestro de etiquetas o en archivos con extensión .cat.
- ***Hscript.exe***.- crea otros archivos que serán usados en el entrenamiento y reconocimiento. Dichos archivos contienen información con descripción binaria del reconocedor, como listas de categorías, etc.

Ya que los archivos han sido seleccionados, el archivo de categorías ha sido creado, y el archivo desc es correcto entonces se hace la selección de los datos, usando los siguientes scripts:

- ***Pickframes.tcl***.- se usa para seleccionar los ejemplos a entrenar. Aquí se genera un archivo llamado pick el cual es usados posteriormente.
- ***Genvec.tcl***.-se usa para crear las características para cada frame a ser entrenado.
- ***Checkvec.exe***.- es para asegurarnos que el archivo de vectores es válido.

Entrenamiento y evaluación

- ***Nntrain.exe***.- usado para entrenar la red sobre el archivo de vectores.
- ***Find_best.tcl***.- se usa para encontrar la mejor iteración de la red usando el conjunto de archivos de desarrollo.
- ***Browse.tcl***.- utilizado para evaluar errores.

Se puede hacer un re-entrenamiento hasta alcanzar el mejor desempeño de todas las redes, aunque algunas veces la primera red tiene el mejor desempeño.

3.2 Ambiente de desarrollo del CSLU-HMM

El ambiente de desarrollo del CSLU-HMM es una colección de bloques construidos con el ánimo de prover al usuario un ambiente de desarrollo e investigación fácil de usar, además de poderoso para la construcción del estado de arte de los HMM, redes neuronales y reconocedores híbridos. Basado en el CSLUShell, el cual usa tcl y tk que nos da un ambiente para ejecutar algunos scripts en dichos lenguajes.

3.2.1 Arquitectura del CSLU-HMM

El CSLU-HMM soporta un ambiente flexible para varias estrategias de modelado. Se debe tener mucho cuidado para diseñar todos estos componentes para poder operarlo eficientemente y de manera consistente, y se le debe dar una atención especial a la modularidad, portabilidad y extensibilidad.

En la *figura 3.1* se presenta la arquitectura del software del CSLU-HMM. Esta es un extensión del shell de CSLU, también usa una gran variedad de módulos pre-existentes para la computación distribuida, procesamiento de señales de voz, operaciones matemáticas y varios módulos que provén un completo ambiente de desarrollo para los HMM.

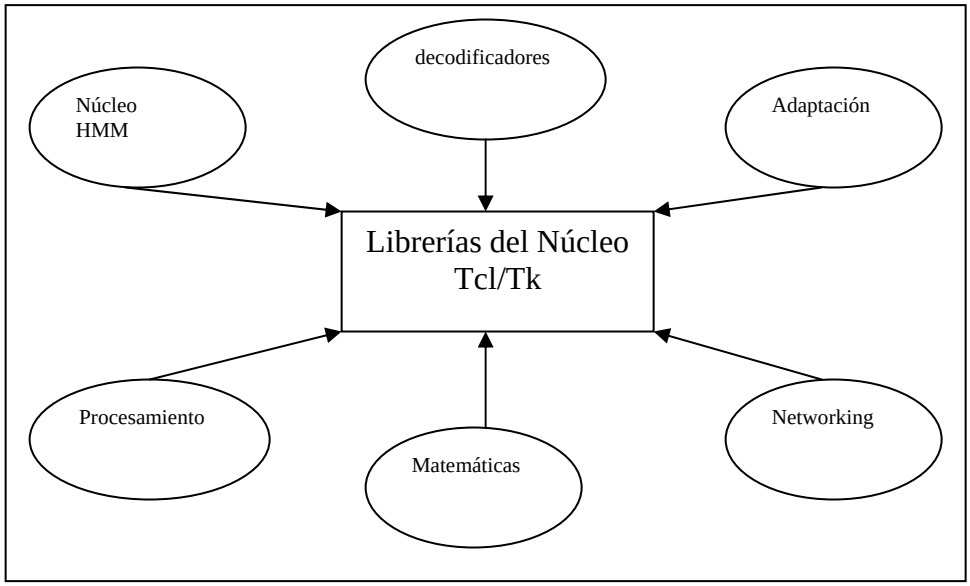

Figura 3.1 Arquitectura del software del CSLU-HMM

3.2.2 El núcleo del CSLU-HMM

La versión actual del software soporta métodos de entrenamiento avanzado y estándar. Para un entrenamiento básico del modelo, las técnicas son las siguientes:

- Modelo de inicialización usando VQ.
- Modelo entrenamiento basado en el algoritmo EM (*expectation/maximization*).

Decodificadores

El ambiente del CSLU-HMM incluye decodificadores de Viterbi para los modelos estándar de HMM, así como para reconocedores híbridos basados en redes neuronales. Ambas arquitecturas soportan el reconocimiento de gramáticas de estado finito.

3.2.3 Herramientas del CSLU-HMM

Los scripts del CSLU-HMM provén un ambiente de desarrollo y usa una interfaz tradicional de lineas de comando. Cabe aclarar que dichos scripts están en tcl, lo cual nos

permite acondicionarlos a nuestras necesidades. En la *figura 3.2* podemos ver la estructura, así como los scripts y cada uno de los pasos para la construcción de un reconocedor basado en HMM.

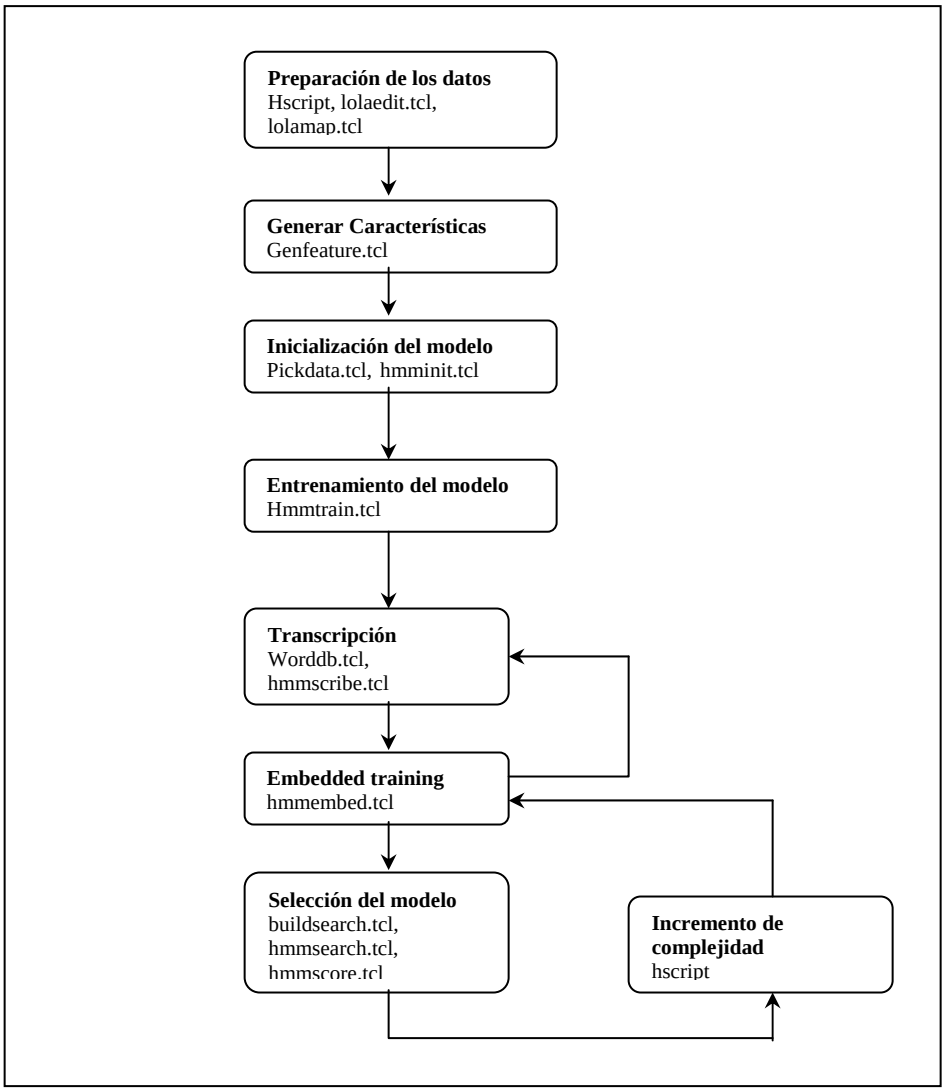

Figura 3.2 Estructura general de un entrenamiento basado en HMM

3.2.4 Preparación de los datos

Es una de las partes más importantes para construir un buen reconocedor. Para esto se necesitan archivos de voz asociados con transcripciones de palabra o fonemas, entonces es generado un MLF(*Master Label File*), el cual contendrá la misma información pero en

un único archivo. Cada uno de los siguientes pasos son muy específicos y los datos deben estar en un formato ya establecido. Ya que tenemos el formato necesitamos saber la arquitectura de nuestro reconocedor. La arquitectura define las unidades básicas usadas durante el reconocimiento. Dependiendo de la aplicación estas unidades pueden ser palabras, sílabas, fonemas o subfonemas. Todo esto es posible usando el lenguaje de configuración del CSLU-HMM. Un ejecutable del CSLU-HMM, *hscript* lee la configuración de dicho modelo y genera un conjunto de modelos base. Estos modelos definirán la arquitectura y los valores de los parámetros iniciales para las probabilidades de transición de los estados [SCHALKWYK2000].

3.2.5 Generación de características

Las características son calculadas para cada archivo de voz. La herramienta ***`genfeature.tcl`*** lee una lista de archivos desde un *master label file* y calcula las características para cada uno de los archivos especificados. Estas características son guardadas en el caché del disco e indexadas para una rápida extracción.

Inicialización del modelo

El proceso de inicialización del modelo está hecho en tres pasos, empezando con las alineaciones obtenidas de los datos inciales etiquetados. La herramienta ***`pickdata.tcl`*** lee la información alineada y selecciona un subconjunto para la incialización y entrenamiento del modelo.

Con los datos seleccionados los parámetros del modelo son inicializados usando la herramienta ***`hmminit.tcl`***. Los segmentos de los datos de una particular clase son cargados en memoria. Cada segmento es entonces cortado en subsegmentos de igual tamaño, dependiendo del número total de estados para el modelo. Los datos son colocados en un particular estado de HMM, y combinados, esto significa que se usa el algoritmo de VQ. La mezcla de covarianzas son un conjunto que es reunido con las covarianzas de los datos. Durante estos pasos los pesos de las mezclas y las probabilidades de los estados de transición no son estimadas.

3.2.6 Entrenamiento del modelo

Durante la inicialización del modelo el parámetro del estado estima una colocación discreta de datos para cada estado en el HMM. Sin embargo esta colocación a menudo no es la óptima. Los segmentos de las características que están cerca de las transiciones de los estados son difíciles de asociar con un único estado. Es decir hay una leve asociación con los estados. El algoritmo de *forward/backward* en contraste con el realineamiento de estados de Viterbi calcula esta asociación probabilística de las característcias contra los estados del HMM.

Estas asociaciones probabilísticas son usadas con el algoritmo EM mencionado anteriormente para además mejorar los parámetros iniciales estimados. Para cada segmento el acumulador de los parámetros de las variables son actualizados. Este acumulador es usado para guardar la contribución de cada vector de datos hacia cada mezcla de cada estado.

La herramienta ***`hmmtrain.tcl`*** encapsula este proceso. Juntos con el modelo de incialización estas dos técnicas de entrenamiento son usadas para formar un modelo inicial el cual es usado como base para el entrenamiento del reconocedor.

3.2.7 Transcripción automática/forced alignment

Los datos etiquetados a mano son suficientes para crear un modelo inicial para reconocedores basados en fonemas (el cual no es nuestro caso). Para construir reconocedores con más exactitud y más robustos se requieren de más datos. Muchos de los corpus contienen transcripciones a nivel de palabra, lo cual puede ser usado para aumentar los datos entrenados ya existentes.

La herramienta ***`hmmscribe.tcl`*** calcula los alineamientos de la palabra, fonema o estado. Las transcripciones de entrada de las palabras son usadas para crear una gramática de estados finita donde cada nodo o estado en la gramática contiene una palabra y sus variantes e pronunciación obtenidas de un léxico de pronunciaciones. Entonces el algoritmo estándar de Viterbi es usado para encontrar la mejor ruta posible a través de la gramática.

El script *hmmscribe.tcl* lee un *master label file*. Para cada palabra en la entrada de transcripciones las variantes de pronunciación son leidas de una base de datos de pronunciaciones. La herramienta worddb.tcl lee el un archivo de texto de pares de palabras y pronunciaciones, y crea una base de datos.

Re-estimación de parámetros

La inicialización y el entrenamiento del modelo usa datos asociados con un modelo particular. Durante estos pasos de entrenamiento se asume que los límites fonéticos so definidos y que no hay interacciones entre los modelos vecinos. La re-estimación de parámetros (hmmembed.tcl) direcciona estos problemas creando un modelo compuesto desde las transcripciones asociadas con el modelo. El modelo es usado para calcular el modelo y los alineamientos, desde el cual los parámetros del modelo son actualizados.

3.2.8 Evaluación

Para establecer el proceso de evaluación primero necesitamos construir una gramática de tareas y una búsqueda asociada con la red usada por el decodificador de *Viterbi*. La herramienta *buildsearch* usas ambos, la especificación de la gramática y las pronunciaciones de las palabras para crear una red de búsqueda. Ya que tenemos la red de búsqueda, el reconocimiento es mejorado usando la herramienta hmmsearch.tcl la cual soporta modelados completos de trifonemas dependientes del contexto, con palabras, fonemas, o alineamientos de nivel de estados. Por omisión sólo la primera mejor respuesta se regresa. Sin embargo, el lattice[1] de palabra es regresado y también está disponible conteniendo múltiples hipótesis.

Cuando todos los archivos han sido procesados, las transcripciones de salida son comparadas con las respuestas de reconocimiento para evaluar el desempeño del reconocedor. La última herramienta utilizada es hmmscore.tcl la cual usa el algoritmo de puntaje NIST (el cual estará explicado más adelante) para comparar las respuestas de reconocimiento con las verdaderas respuestas obtenidas.

[1] Búsqueda basada en refinamiento sucesivo del modelo del lenguaje. Indica además cuales palabras fueron reconocidas para ser habladas en esos intervalos .

Una vez dadas las bases sobre redes neuronales y modelos ocultos de Markov, y el ambiente de desarrollo para ambas técnicas, continuaremos en el siguiente capítulo con los diseños de los experimentos y evaluación de los resultados.