

Procesamiento del Lenguaje Natural y Análisis de Sentimientos (Parte 1)

Carlos Gómez-Rodríguez y Eugenio Martínez-Cámara

Octubre 2018

CAEPIA 2018, Granada



UNIVERSIDADE DA CORUÑA



UNIVERSIDAD
DE GRANADA

Índice

Introducción

Representaciones vectoriales del lenguaje

Etiquetación (part-of-speech tagging)

Análisis sintáctico

Análisis de sentimientos

Outline

Introducción

Representaciones vectoriales del lenguaje

Etiquetación (part-of-speech tagging)

Análisis sintáctico

Análisis de sentimientos

Procesamiento del Lenguaje Natural

- ▶ Desarrollar programas informáticos que puedan procesar (comprender y/o generar) lenguaje humano
- ▶ Aplicaciones: traducción automática, resumen automático, extracción de información, búsqueda de respuestas, sistemas de diálogo/chatbots, análisis de sentimientos...

Análisis de sentimientos

- ▶ Extraer opiniones o sentimientos expresados en textos
- ▶ Incluye tareas como:
 - ▶ **Clasificar la polaridad.**
'Love my new phone! Samsung Galaxy Note 3!'
 - ▶ **Extraer evaluación de aspectos.**
'I really like the look of the new HTC One. Hardware is pretty beautiful, fast and great battery. But that camera :('
 - ▶ **Identificar relaciones entre productos.**
'Gotta say, loving the battery on this Sony Xperia Z1 compact. My old iPhone would have died twice over by now. Great phone too. #cool'
 - ▶ **Clasificación temática.**
'The HTC One E8 is a hit in Asia with 50 000 units sold in 15 minutes
HTC needed some good news on the sales front, bit.ly1so0Ret'
 - ▶ ...

Uso de información lingüística

- ▶ Para procesar con éxito textos en lenguaje natural, necesitaremos extraer información lingüística:
 - ▶ Categoría gramatical de cada palabra (verbo, sustantivo...): **Part-of-Speech (PoS) tagging**
 - ▶ Función que juega cada palabra en la oración (sujeto, objeto...): **Análisis sintáctico (parsing)**
 - ▶ Tagging, parsing y análisis de sentimientos se llevarán a cabo con aprendizaje automático (modelos neuronales)
 - ▶ Necesitamos **representaciones vectoriales del lenguaje**

Uso de información lingüística

Por lo tanto, un posible “pipeline” es:

1. Transformamos las palabras del texto en vectores
(representaciones vectoriales/word embeddings)
2. Etiquetamos cada palabra con su categoría gramatical
(part-of-speech (PoS) tagging)
3. Obtenemos las dependencias sintácticas entre palabras
(dependency parsing)
4. Utilizamos todo ello como características para nuestra tarea final (análisis de sentimiento)

Outline

Introducción

Representaciones vectoriales del lenguaje

Etiquetación (part-of-speech tagging)

Análisis sintáctico

Análisis de sentimientos

Representaciones vectoriales del lenguaje

- ▶ La mayoría de los sistemas de PLN hasta \approx 2013 trataban las palabras como símbolos atómicos.
- ▶ En términos de espacio vectorial para aprendizaje automático, esto se corresponde con una representación *one-hot*:
- ▶ $\text{perro} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]$, $\text{gato} = [0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$
- ▶ Inconvenientes:
 - ▶ Muchas arquitecturas neuronales trabajan mejor con vectores *densos* de números reales
 - ▶ No hay noción de *similitud*:
 - ▶ $\text{avión} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0]$
 - ▶ $\text{aeroplano} = [0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$
 - ▶ El modelo no tiene modo de saber que estas dos palabras tienen más relación que “avión” y “gato”

Representaciones vectoriales del lenguaje

- ▶ El objetivo de las representaciones vectoriales densas del lenguaje (*word embeddings*) es:
 - ▶ Proporcionar una representación **compacta y densa** (en comparación con 1-hot)
 - ▶ Modelar la **similitud** entre palabras:
 - ▶ avión = [0.92 0.01 0.54]
 - ▶ aeroplano = [0.91 0.02 0.53]
 - ▶ gato = [0.24 0.73 0.22]

Representaciones vectoriales del lenguaje

- ▶ Los modelos de representaciones vectoriales densas del lenguaje se basan en el contexto.
- ▶ “You shall know a word by the company it keeps” (J.R. Firth, 1957)
 - ▶ ...cuando un avión está volando, la presión del aire en la parte superior del ala...
 - ▶ ...el avión en el que viajaba tuvo que hacer un aterrizaje de emergencia...
 - ▶ ...reposan las piezas inacabadas de un avión Antónov An-225 en un hangar...
 - ▶ ...aunque el avión es el medio de transporte más seguro, son muchos los pasajeros que tienen miedo a volar...
- ▶ Estas palabras representarán a “avión”.
- ▶ Una palabra que aparezca en contextos similares (volando, aire, ala, viajaba, aterrizaje, hangar, transporte...) seguramente se parecerá mucho a “avión”.

Representaciones vectoriales del lenguaje: enfoques

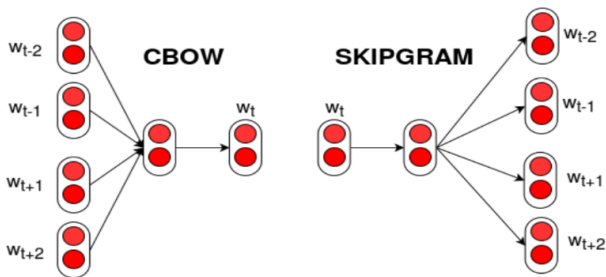
- ▶ ¿Cómo obtener estas representaciones basadas en el contexto?
 - ▶ Mediante factorización de **matrices** palabra-contexto (LDA, GloVe)
 - ▶ Mediante **redes neuronales** (Word2vec, ELMo)

Representaciones vectoriales del lenguaje:

Word2Vec

- ▶ Dos modelos de redes de neuronas de dos capas:
 - ▶ Entrada: vectores 1-hot
 - ▶ Capa oculta: lineal
 - ▶ Capa de salida: clasificador softmax
- ▶ Se entrenan en un corpus grande de textos.
- ▶ Dos modelos:
 - ▶ **CBOW** (Conditional Bag of Words): predecir cada palabra a partir del contexto
 - ▶ **Skip-gram**: predecir el contexto a partir de la palabra
- ▶ Los **pesos de la capa oculta** (filas de la matriz de pesos) son los vectores word2vec (word embeddings)

Representaciones vectoriales del lenguaje: Word2Vec



$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_t | w_{t+j})$$

$$\frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$$

Imagen CC-BY por Aelu13

Propiedades de Word2Vec

- ▶ Word2Vec captura **relaciones semánticas** entre los términos
 - ▶ $\text{rey} - \text{hombre} + \text{mujer} \approx \text{reina}$
 - ▶ $\text{Italia} - \text{Madrid} + \text{España} \approx \text{Roma}$

Representaciones vectoriales del lenguaje:

inicialización aleatoria

- ▶ Otra opción: **inicialización aleatoria**
 - ▶ Usar una primera capa en nuestra tarea que transforme vectores 1-hot en embeddings
 - ▶ Como añadir a nuestro modelo la capa oculta de word2vec
 - ▶ Las “embeddings” se inicializan aleatoriamente para cada palabra del vocabulario (son los pesos de dicha capa)
 - ▶ Así, el entrenamiento de esas representaciones vectoriales se integra con el de la propia tarea
 - ▶ Ventaja: entrenamos unas “embeddings” **específicas para nuestra tarea**
 - ▶ Inconveniente: normalmente, **menos datos de entrenamiento**

Representaciones vectoriales del lenguaje: aplicación

- ▶ Estas técnicas de representación vectorial se pueden aplicar a otras unidades lingüísticas (PoS tags, caracteres, etc.)
- ▶ Nos permiten representar el lenguaje con vectores cortos, densos y continuos
- ▶ Útiles para usar lenguaje natural como **entrada a modelos neuronales** y de aprendizaje profundo
- ▶ Se pueden descargar representaciones **preentrenadas** (Word2vec, GloVe) o entrenar propias, genéricas o específicas para una tarea (con entrenamiento integrado en el de la tarea)
- ▶ Mediante representaciones vectoriales + aprendizaje profundo, se ha **mejorado la precisión** de la mayoría de las tareas de NLP (análisis morfológico, sintáctico, traducción automática, análisis de sentimientos, etc.)

Outline

Introducción

Representaciones vectoriales del lenguaje

Etiquetación (part-of-speech tagging)

Análisis sintáctico

Análisis de sentimientos

Part-of-speech tagging

- ▶ Objetivo: asignar categorías gramaticales (PoS tags) a las palabras.
- ▶ Ambigüedad:
 - ▶ El/**DET** sobre/**N** está/**V** sobre/**PRP** la/**DET** mesa/**N**
 - ▶ Buffalo/**ADJ** buffalo/**N** Buffalo/**ADJ** buffalo/**N** buffalo/**V**
buffalo/**V** Buffalo/**ADJ** buffalo/**N**

Part-of-speech tagging

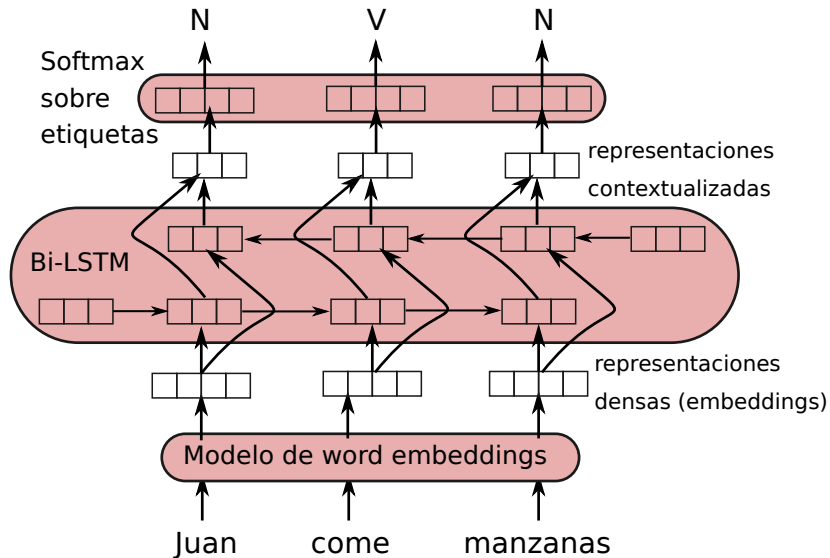
- ▶ Objetivo: asignar categorías gramaticales (PoS tags) a las palabras.
- ▶ Ambigüedad:
 - ▶ El/**DET** sobre/**N** está/**V** sobre/**PRP** la/**DET** mesa/**N**
 - ▶ Buffalo/**ADJ** buffalo/**N** Buffalo/**ADJ** buffalo/**N** buffalo/**V**
buffalo/**V** Buffalo/**ADJ** buffalo/**N**
 - ▶ (Buffalo bisons Buffalo bisons bully bully Buffalo bisons)
 - ▶ (The buffalo from Buffalo who are buffaloed by buffalo from Buffalo, in turn buffalo other buffalo from Buffalo)
- ▶ Aproximación usual: entrenar modelos sobre corpus etiquetados.

Part-of-speech tagging

Modelos típicos:

- ▶ Modelos de Markov ocultos (hidden Markov models)
- ▶ Modelos de máxima entropía (MaxEnt)
- ▶ Conditional random fields (CRF)
- ▶ Modelos neuronales: LSTM+CRF, LSTM+Softmax.
- ▶ “To a first approximation, the de facto consensus in NLP in 2017 is that no matter what the task, you throw a BiLSTM at it, with attention if you need information flow” (Christopher D. Manning, Stanford, 2017)

Part-of-speech tagging con Bi-LSTM y Softmax



Outline

Introducción

Representaciones vectoriales del lenguaje

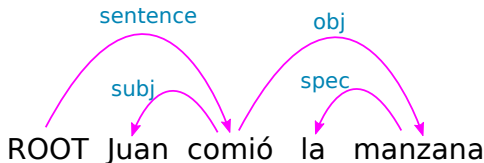
Etiquetación (part-of-speech tagging)

Análisis sintáctico

Análisis de sentimientos

Análisis sintáctico de dependencias

- ▶ Análisis sintáctico (parsing): Obtener la **estructura sintáctica** de las oraciones.
- ▶ Análisis sintáctico **de dependencias**: Expresada mediante **dependencias**:

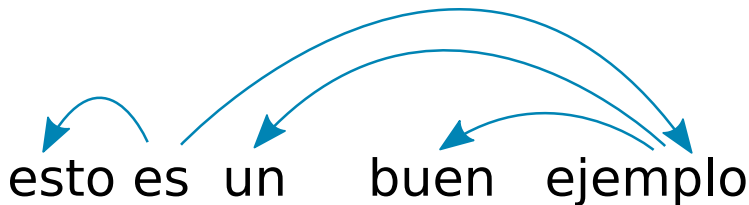


- Información crucial para aplicaciones de PLN como el análisis de sentimientos.

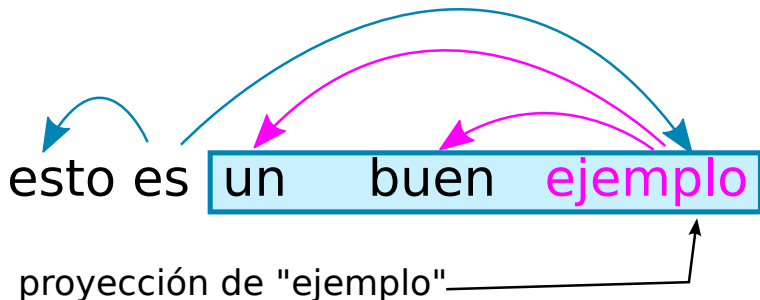
Análisis de dependencias proyectivo

esto es un buen ejemplo

Análisis de dependencias proyectivo



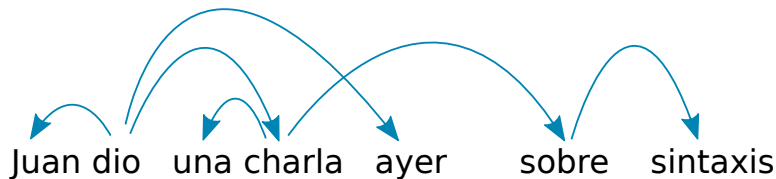
Análisis de dependencias proyectivo



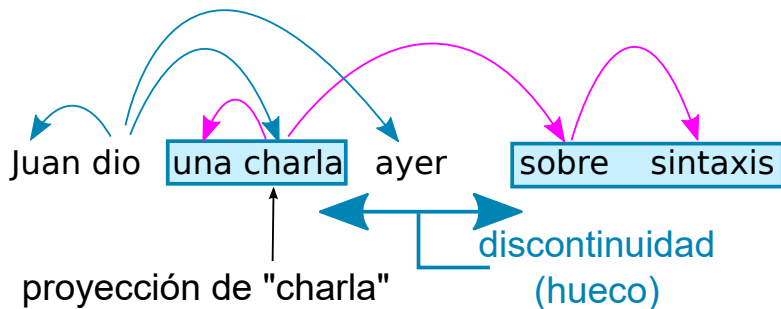
Análisis de dependencias no proyectivo

Juan dio una charla ayer sobre sintaxis

Análisis de dependencias no proyectivo



Análisis de dependencias no proyectivo



Principales enfoques

No se basan en gramática (dirigidos por los datos, **data-driven**)
Robustez (salida para cualquier entrada), **desambiguación** (salida única)

- ▶ Análisis **basado en transiciones**
 - ▶ Basado en una máquina de estados no determinista + modelo de puntuación para secuencias de transiciones
 - ▶ Inferencia aproximada: $O(n)$ análisis proyectivo, $O(n)$ suavemente no proyectivo, $O(n^2)$ no proyectivo ($n =$ longitud de la oración)
 - ▶ Inferencia exacta: $O(n^3)$ análisis proyectivo, $O(n^4)$ suavemente no proyectivo
- ▶ Análisis **basado en grafos**
 - ▶ Basado en puntuar fragmentos de grafo/árbol
 - ▶ Inferencia exacta: $O(n^3)$ análisis proyectivo, $O(n^2)$ no proyectivo (pero con funciones de puntuación limitadas)

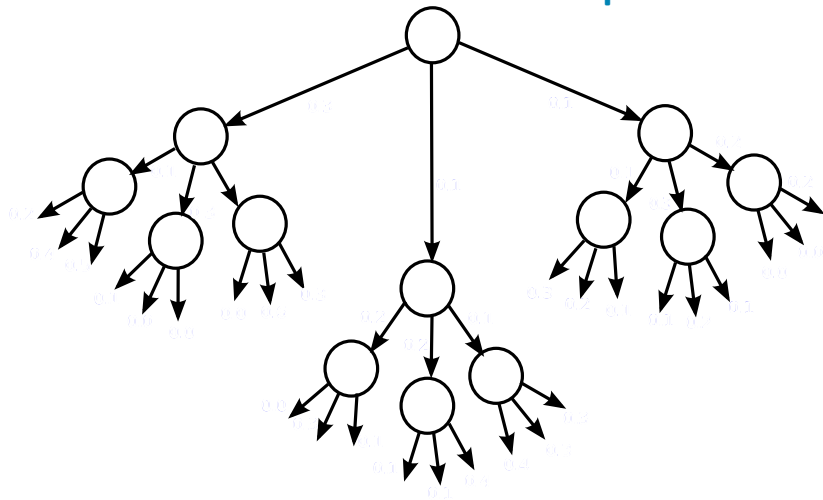
Análisis basado en transiciones: Elementos

- ▶ Máquina de estados no determinista
- ▶ Modelo de puntuación para secuencias de transiciones (probabilístico o de machine learning)
- ▶ Búsqueda de la mejor secuencia

Análisis basado en transiciones: Elementos

- ▶ Máquina de estados no determinista
- ▶ Modelo de puntuación para secuencias de transiciones (probabilístico o de machine learning)
- ▶ Búsqueda de la mejor secuencia
 - ▶ Búsqueda voraz (greedy search)
 - ▶ Búsqueda en haz (beam search)
 - ▶ Programación dinámica (inferencia exacta)

Análisis basado en transiciones: Búsqueda



Análisis basado en transiciones: Máquina de estados

Sistemas de transiciones (autómatas/máquinas de estados)

proyectivos:

- ▶ Múltiples algoritmos sencillos, lineales, que usan una pila (arc-standard, arc-eager, arc-hybrid, planar)

Sistemas de transiciones no proyectivos:

- ▶ Intercambios de orden en los nodos (Nivre swap)
- ▶ No quitar los nodos de la memoria (Covington)

Sistemas de transiciones suavemente no proyectivos:

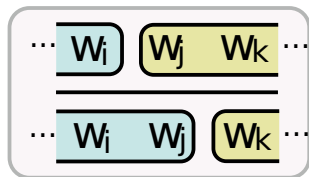
- ▶ Transiciones de arcos no adyacentes (Attardi)
- ▶ Dos pilas (2-Planar)

Ejemplo proyectivo: analizador Planar

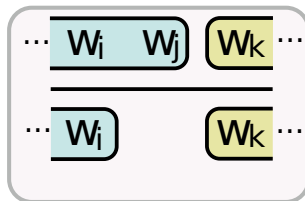
- ▶ Variante del analizador proyectivo “arc-eager” de Nivre (2003)
- ▶ Analizador basado en transiciones de izquierda a derecha, lineal, con
 - ▶ **Buffer** con palabras de entrada no procesadas
 - ▶ **Pila** con palabras de entrada parcialmente procesadas (inicialmente vacía)
 - ▶ Cuatro transiciones: Shift, Reduce, Left-Arc, Right-Arc

Analizador Planar: transiciones

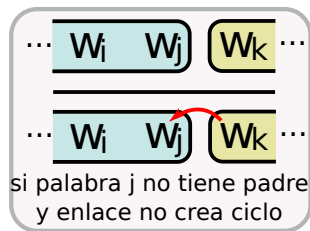
Shift



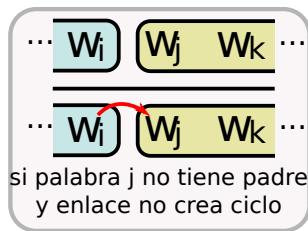
Reduce



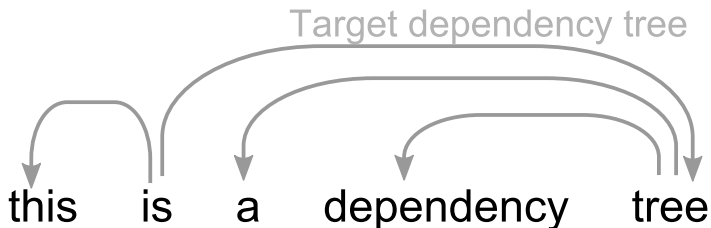
Left-Arc




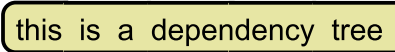
Right-Arc



Analizador Planar: ejemplo




Stack

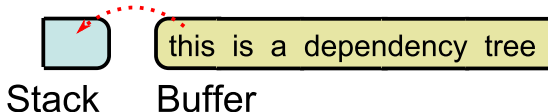

Buffer

Done:

Action:

Analizador Planar: ejemplo

this is a dependency tree



Done:

Action: **Shift (this)**

Analizador Planar: ejemplo

 this is a dependency tree

this

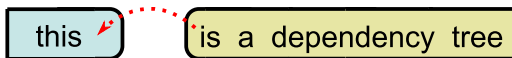
is a dependency tree

Done: Shift (this)

Action: **Left-Arc** (this, is)

Analizador Planar: ejemplo

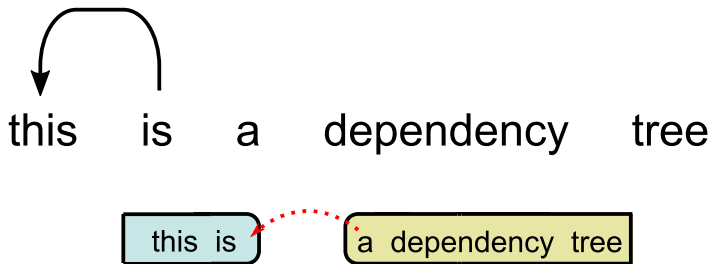
this is a dependency tree

A black curved arrow originates from the word 'is' and points back to the word 'this', representing a dependency arc.

Done: Left-Arc (this, is)

Action: **Shift** (is)

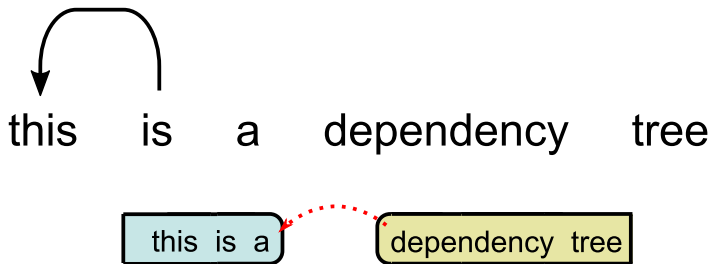
Analizador Planar: ejemplo



Done: Shift (is)

Action: **Shift (a)**

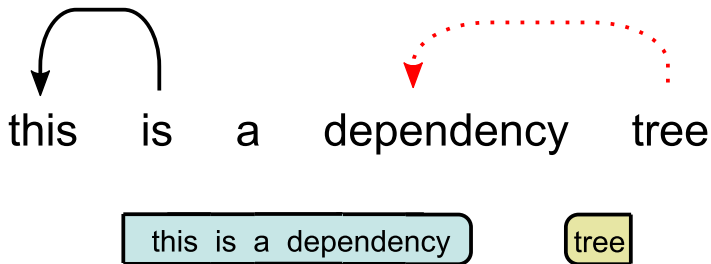
Analizador Planar: ejemplo



Done: Shift (a)

Action: **Shift (dependency)**

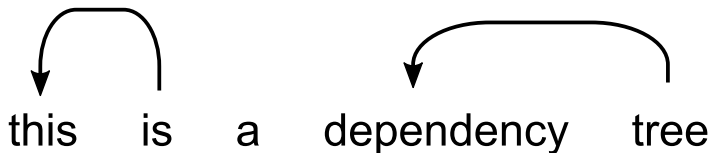
Analizador Planar: ejemplo



Done: Shift (dependency)

Action: **Left-Arc (dependency, tree)**

Analizador Planar: ejemplo



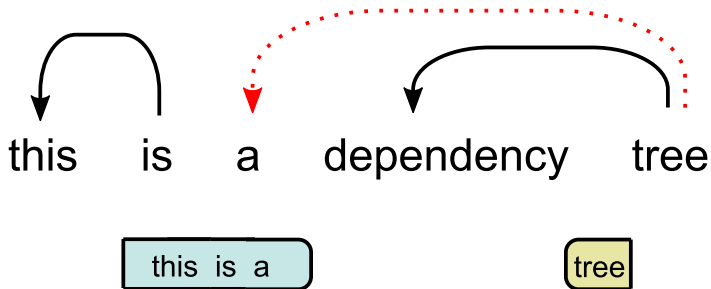
this is a dependency

tree

Done: Left-Arc (dependency , tree)

Action: **Reduce (dependency)**

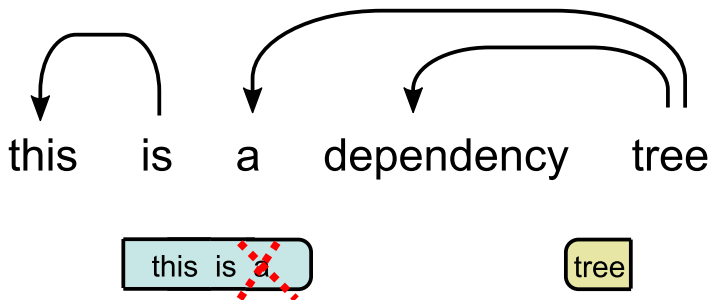
Analizador Planar: ejemplo



Done: Reduce (dependency)

Action: **Left-Arc (a , tree)**

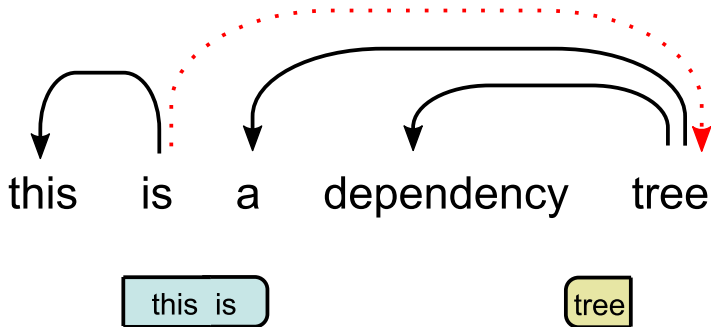
Analizador Planar: ejemplo



Done: Left-Arc (a , tree)

Action: **Reduce** (a)

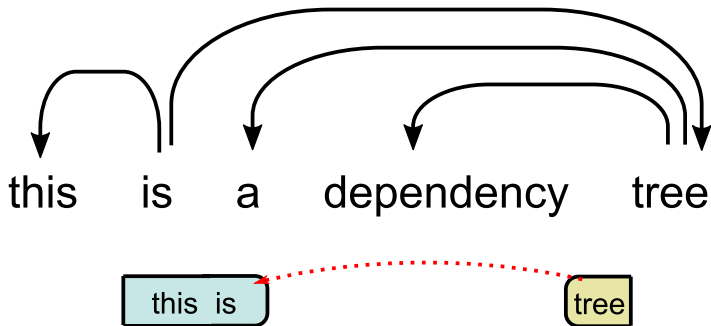
Analizador Planar: ejemplo



Done: Reduce (a)

Action: **Right-Arc** (is , tree)

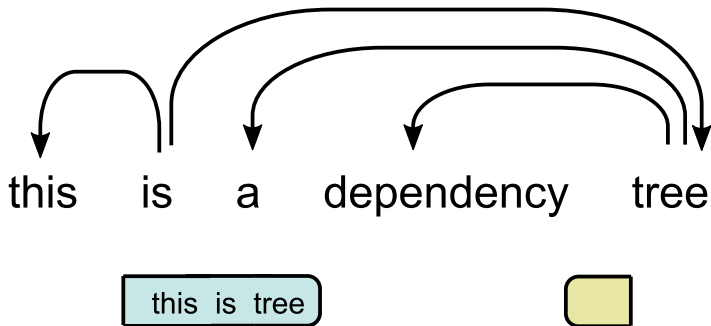
Analizador Planar: ejemplo



Done: Right-Arc (is , tree)

Action: **Shift (tree)**

Analizador Planar: ejemplo



Done: Shift (tree)

Action: **Done!**

Ejemplo suavemente no proyectivo: analizador 2-Planar

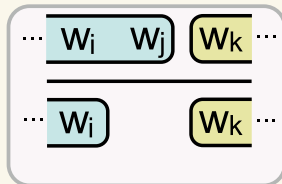
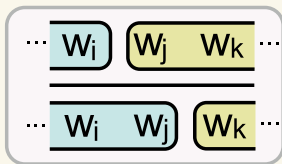
- ▶ Generalización del analizador Planar para ampliar su cobertura a oraciones no proyectivas
- ▶ Cubre árboles **2-planares** (divisibles en dos subgrafos, planos, sin dependencias cruzadas entre planos)
- ▶ Funcionamiento:
 - ▶ **Dos pilas** (una por plano) en lugar de una
 - ▶ Las transiciones Shift insertan palabras en *ambas* pilas
 - ▶ Reduce, Left-Arc, Right-Arc trabajan con una sola pila (pila *activa*)
 - ▶ Una transición adicional *Switch* cambia qué pila está activa

Analizador 2-Planar: transiciones

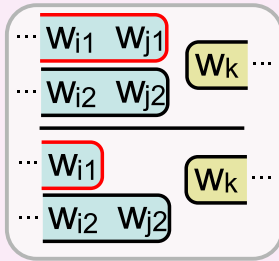
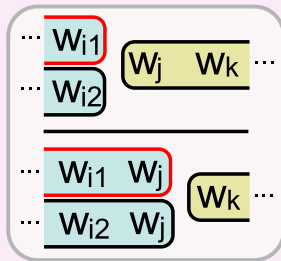
Shift

Reduce

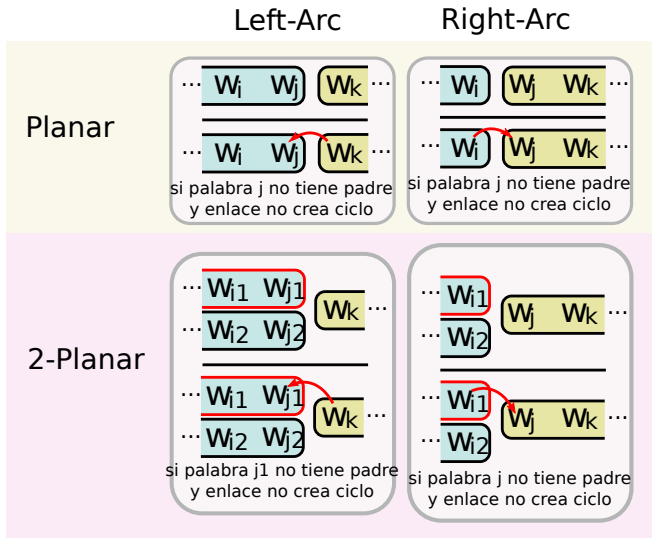
Planar



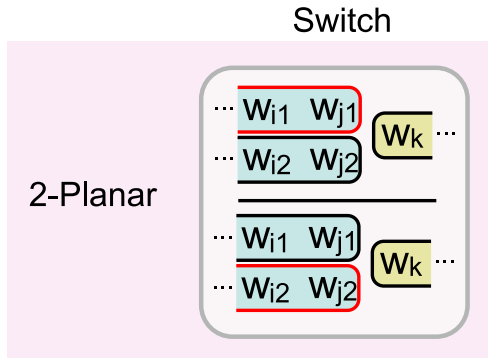
2-Planar



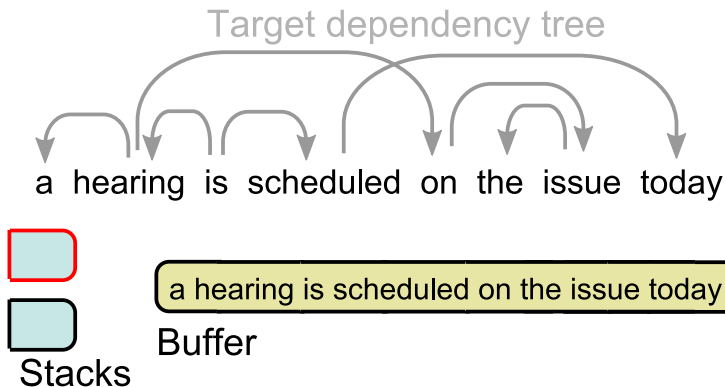
Analizador 2-Planar: transiciones



Analizador 2-Planar: transiciones



Analizador 2-Planar: ejemplo

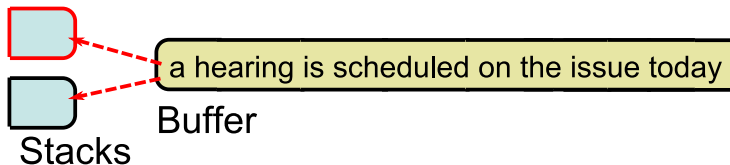


Done:

Action:

Analizador 2-Planar: ejemplo

a hearing is scheduled on the issue today



Done:

Action: **Shift (a)**

Analizador 2-Planar: ejemplo



a hearing is scheduled on the issue today

a

a

Stacks

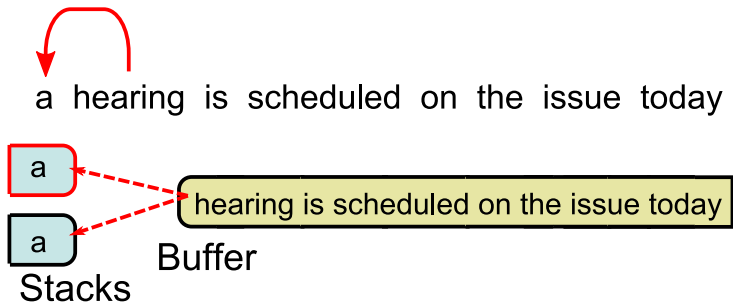
hearing is scheduled on the issue today

Buffer

Done: Shift (a)

Action: **Left-Arc (a , hearing)**

Analizador 2-Planar: ejemplo



Done: Left-Arc (a , hearing)

Action: **Shift (hearing)**

Analizador 2-Planar: ejemplo



a hearing is scheduled on the issue today

a hearing

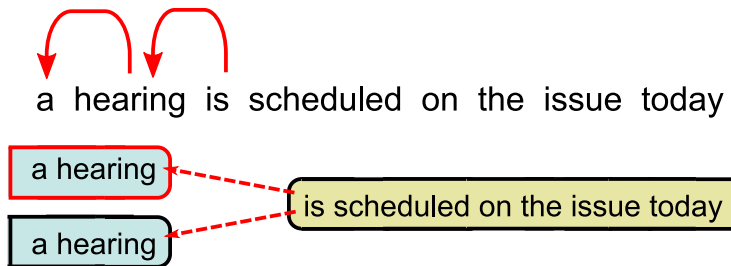
is scheduled on the issue today

a hearing

Done: Shift (hearing)

Action: Left-Arc (hearing , is)

Analizador 2-Planar: ejemplo



Done: Left-Arc (hearing , is)

Action: **Shift (is)**

Analizador 2-Planar: ejemplo



a hearing is scheduled on the issue today

a hearing is

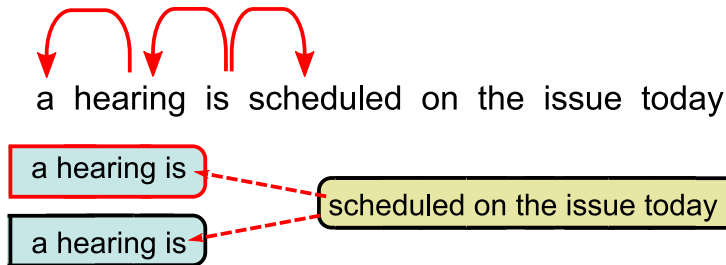
scheduled on the issue today

a hearing is

Done: Shift (is)

Action: **Right-Arc (is,scheduled)**

Analizador 2-Planar: ejemplo



Done: Right-Arc (is,scheduled)

Action: **Shift (scheduled)**

Analizador 2-Planar: ejemplo



a hearing is scheduled on the issue today

~~a hearing is scheduled~~

on the issue today

a hearing is scheduled

Done: Shift (scheduled)

Action: Reduce (scheduled)

Analizador 2-Planar: ejemplo



a hearing is scheduled on the issue today

a hearing is

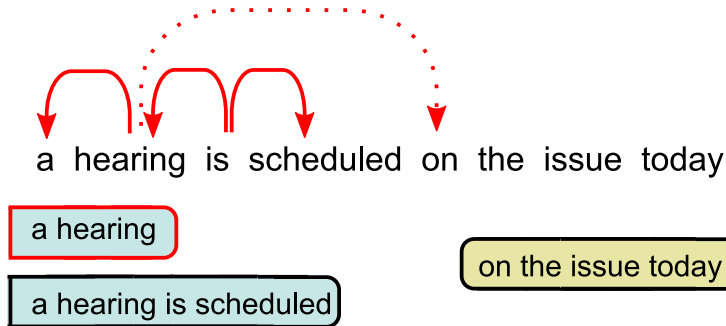
on the issue today

a hearing is scheduled

Done: Reduce (scheduled)

Action: Reduce (is)

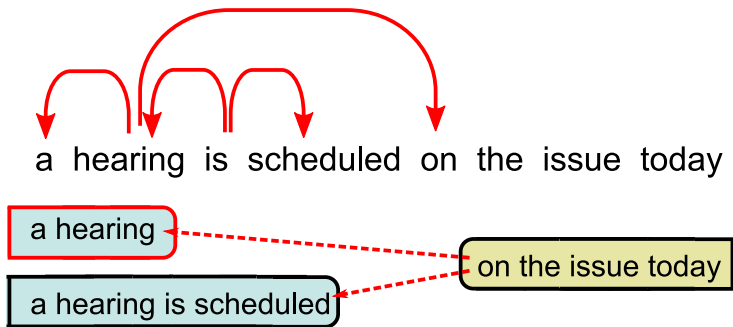
Analizador 2-Planar: ejemplo



Done: Reduce (is)

Action: **Right-Arc (hearing,on)**

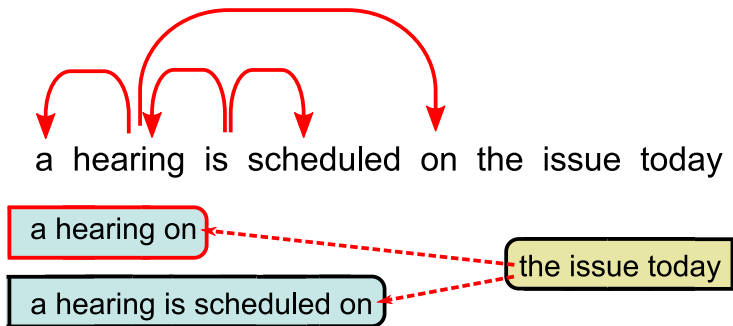
Analizador 2-Planar: ejemplo



Done: Right-Arc (hearing,on)

Action: **Shift** (on)

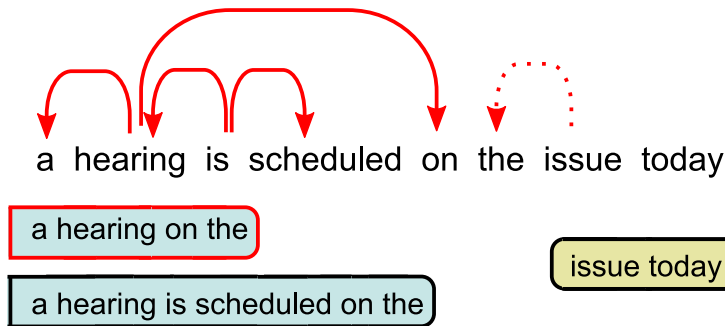
Analizador 2-Planar: ejemplo



Done: Shift (on)

Action: **Shift (the)**

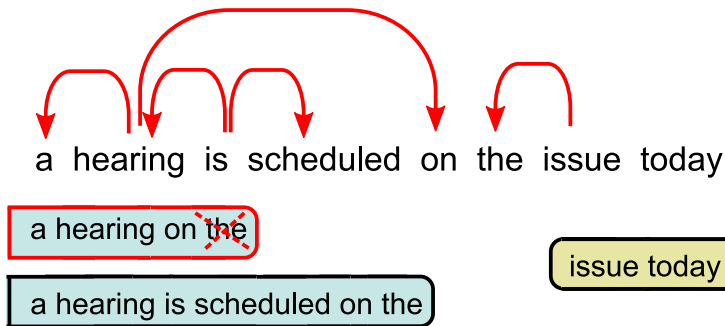
Analizador 2-Planar: ejemplo



Done: Shift (the)

Action: **Left-Arc** (the , issue)

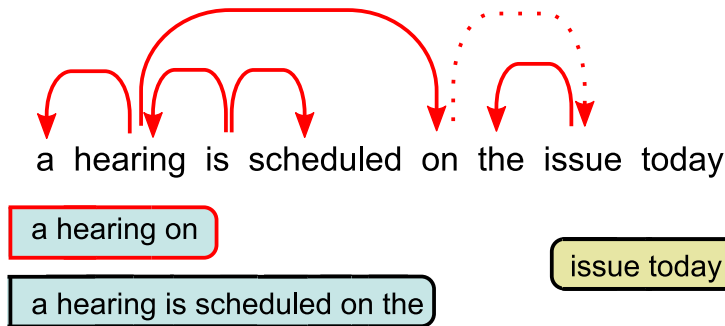
Analizador 2-Planar: ejemplo



Done: Left-Arc (the , issue)

Action: Reduce (the)

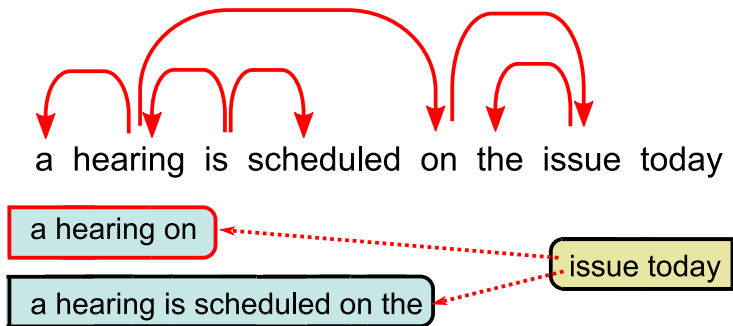
Analizador 2-Planar: ejemplo



Done: Reduce (the)

Action: **Right-Arc** (on , issue)

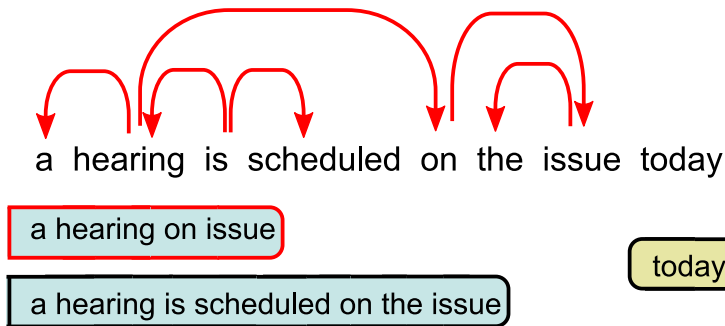
Analizador 2-Planar: ejemplo



Done: Right-Arc (on , issue)

Action: **Shift (issue)**

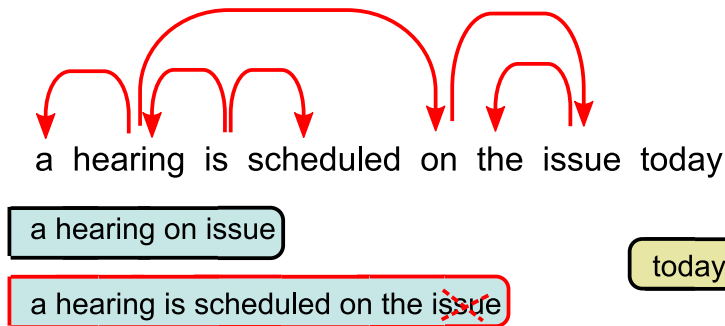
Analizador 2-Planar: ejemplo



Done: Shift (issue)

Action: **Switch**

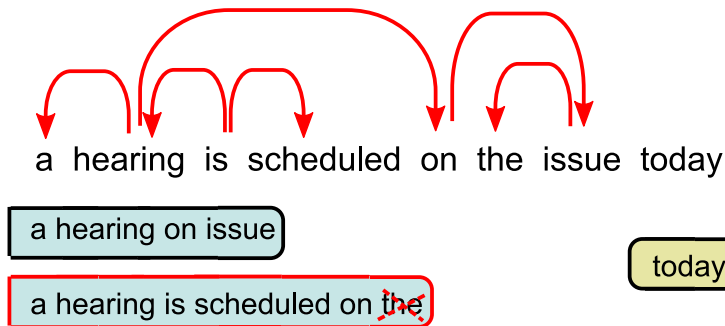
Analizador 2-Planar: ejemplo



Done: Switch

Action: Reduce (issue)

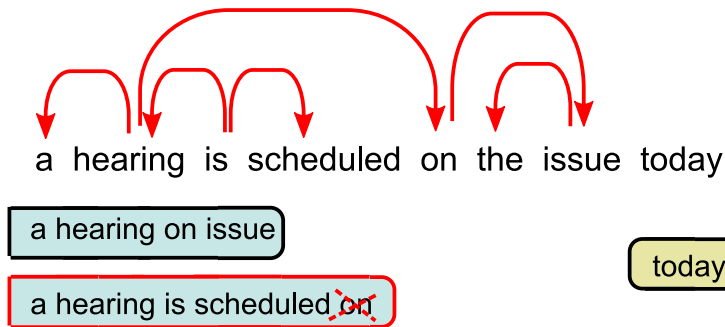
Analizador 2-Planar: ejemplo



Done: Reducie (issue)

Action: Reduce (the)

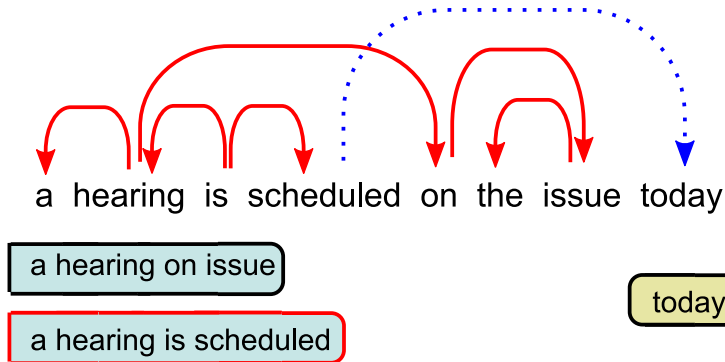
Analizador 2-Planar: ejemplo



Done: Reduce (the)

Action: Reduce (on)

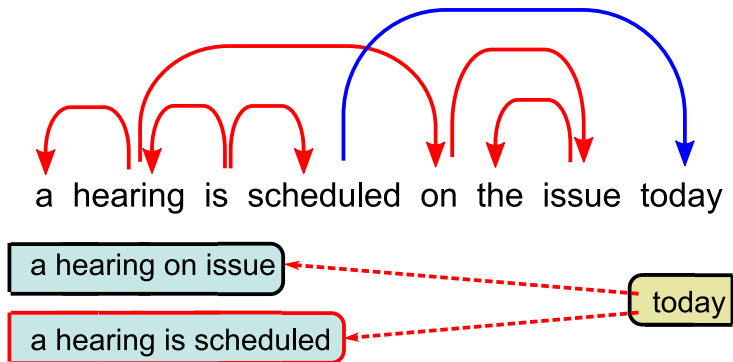
Analizador 2-Planar: ejemplo



Done: Reduce (on)

Action: Right-Arc (scheduled,today)

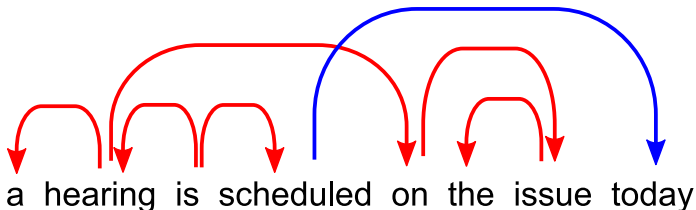
Analizador 2-Planar: ejemplo



Done: Right-Arc (scheduled, today)

Action: **Shift** (today)

Analizador 2-Planar: ejemplo



a hearing on issue today

a hearing is scheduled today



Done: Shift (today)

Action: **Done!**

Complejidad computacional de Planar y 2-Planar

Planar y 2-Planar funcionan en tiempo **lineal** ($O(n)$) respecto a la longitud de la oración n :

- ▶ Planar: cada palabra objeto de como mucho un Shift, un Arc (como dependiente) y un Reduce
 - ▶ \Rightarrow Número de transiciones $\leq 3n$
- ▶ 2-Planar: cada palabra objeto de como mucho un Shift, un Arc (como dependiente) y dos Reduce (uno por pila)
- ▶ A mayores, entre cada par de estas transiciones podría ir un Switch
 - ▶ \Rightarrow Número de transiciones $\leq 8n$ ($4n$ otras + $4n$ Switch)

Los analizadores no proyectivos puros abandonan el tiempo lineal

Entrenamiento de analizadores basados en transiciones

- ▶ Se parte de un corpus (**treebank**) de entrenamiento con el árbol correcto para cada oración
- ▶ Se entrena un clasificador o puntuador para aproximar un **oráculo**: (árbol correcto, configuración) → conjunto de transiciones
- ▶ La salida de este modelo para cada transición puede ser continua (puntuador) o discreta (clasificador, nos dice si debemos tomar la transición (1) o no (0))
- ▶ El oráculo puede ser estático o dinámico

Entrenamiento de analizadores basados en transiciones

► Oráculo estático

- Definido sólo en configuraciones de la computación (secuencia de transiciones) canónica obtenida del árbol correcto.
- Ignora la ambigüedad espuria (siempre devuelve exactamente una transición)
- No se hace entrenamiento en configuraciones erróneas

► Oráculo dinámico

- Definido en todas las configuraciones
- Soporta ambigüedad espuria (puede devolver varias transiciones)
- Devuelve transiciones que conducen a la **mínima pérdida** (no necesariamente pérdida 0) con respecto al árbol correcto

Arquitecturas de aprendizaje para analizadores basados en transiciones

Arquitectura típica pre- “Deep Learning”:

- ▶ Codificación de características en vectores “1-hot”
- ▶ Se define un modelo de **características** para representar con un vector cada estado del algoritmo
 - ▶ p.ej. “las dos primeras palabras de la pila y la primera del buffer”
 - ▶ Los vectores “1-hot” de todas ellas se concatenan para formar un vector más grande
 - ▶ Se entrena el modelo para cada transición. Entrada: representación de un estado. Salida deseada: 1 ó 0 según si la transición es deseable o no.
- ▶ Modelos más utilizados: SVMs (MaltParser, Nivre, 2006), perceptrón estructurado (ZPar, Zhang & Nivre, 2011)

Arquitecturas de aprendizaje para analizadores basados en transiciones

Arquitectura típica pre- “Deep Learning”:

- ▶ Definir un buen modelo de características es clave para conseguir buenos resultados
 - ▶ Modelos de características enormemente **complejos** (decenas de características que pueden incluir palabras, PoS tags, lemas, tipos de dependencia, etc.)
 - ▶ Considerable consumo de memoria (p.ej. 10 palabras, tamaño de vocabulario 50 000 → vector de 500 000 elementos)
 - ▶ Búsqueda manual de modelos que den buen resultado para un idioma dado: “**feature engineering**”
 - ▶ Se usaba siempre búsqueda voraz o en haz. La inferencia exacta sólo es posible para modelos de características muy simples

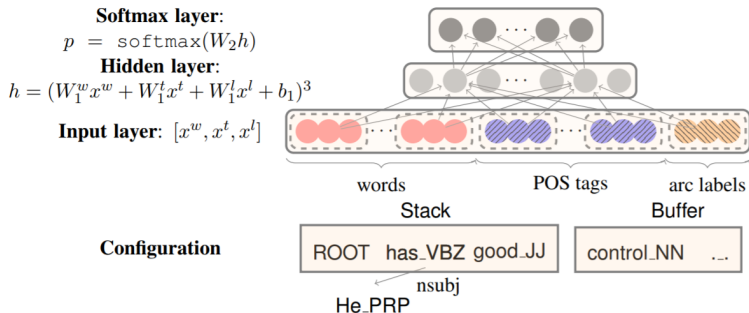
Arquitecturas de aprendizaje para analizadores basados en transiciones

Arquitectura típica con “Deep Learning”:

- ▶ Codificación de características con vectores de reales (**embeddings**), como se ha visto antes
- ▶ Se usan embeddings tanto para palabras como para PoS tags, etiquetas de dependencia, etc.
- ▶ Arquitecturas más populares: redes **feed-forward** (Stanford CoreNLP, Chen & Manning, 2014), **BiLSTMs** (BIST Parser, Kiperwasser & Goldberg, 2016), Stack LSTMs (Kuncoro et al, 2016)
- ▶ Modelos más potentes + representaciones vectoriales que tienen en cuenta el contexto (BiLSTMs): hacen falta menos características
 - ▶ Fin del “feature engineering” (pero principio del “**hyperparameter engineering**”)
 - ▶ La inferencia exacta se vuelve posible (Gómez-Rodríguez et al, 2018)

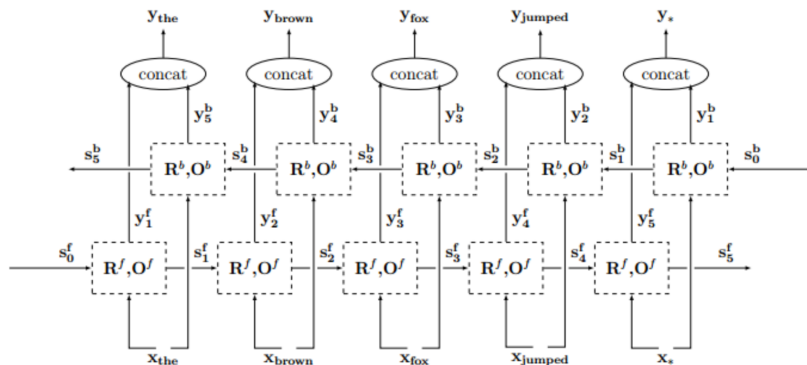
Arquitecturas de aprendizaje para analizadores basados en transiciones

Arquitectura feed-forward (imagen de Chen & Manning, 2016)



Arquitecturas de aprendizaje para analizadores basados en transiciones

Arquitectura de BiLSTM (imagen de Kiperwasser & Goldberg, 2016)



Arquitecturas de aprendizaje para analizadores basados en transiciones

Arquitectura típica con “Deep Learning”:

- ▶ Codificación de características con vectores de reales (**embeddings**), como se ha visto antes
- ▶ Se usan embeddings tanto para palabras como para PoS tags, etiquetas de dependencia, etc.
- ▶ Arquitecturas más populares: redes **feed-forward** (Stanford CoreNLP, Chen & Manning, 2014), **BiLSTMs** (BIST Parser, Kiperwasser & Goldberg, 2016), Stack LSTMs (Kuncoro et al, 2016)
- ▶ Modelos más potentes + representaciones vectoriales que tienen en cuenta el contexto (BiLSTMs): hacen falta menos características
 - ▶ Fin del “feature engineering” (pero principio del “**hyperparameter engineering**”)
 - ▶ La inferencia exacta se vuelve posible (Gómez-Rodríguez et al, 2018)

Estado del arte actual

- ▶ Analizadores basados en grafos y en transiciones prácticamente obtienen la misma precisión
- ▶ Precisión (LAS) en el English Penn Treebank cercana al 95%
- ▶ **Desafíos:**
 - ▶ Idiomas altamente no proyectivos (alemán...)
 - ▶ Idiomas con morfología rica (árabe, turco...)
 - ▶ Idiomas con pocos datos de entrenamiento (kazajo, uzbeko...)
 - ▶ Texto ruidoso (tuits...)
 - ▶ Velocidad de los algoritmos de análisis sintáctico

Outline

Introducción

Representaciones vectoriales del lenguaje

Etiquetación (part-of-speech tagging)

Análisis sintáctico

Análisis de sentimientos