Procesamiento de Lenguaje Escrito

Ana X. Ezquerro

ana.ezquerro@udc.es, GitHub

Grado en Ciencia e Ingeniería de Datos
Universidad de A Coruña (UDC)

Curso 2021-2022

Tabla de Contenidos

Tema 1: Lenguaje Estructurado

1.1. Clasificación de Chomsky

- **Gramáticas Tipo 0**: Gramáticas con estructura de frase o reconocibles por una *Máquina de Turing*.

$$\alpha \to \beta, \quad \alpha \in (\mathbf{N} \cup \mathbf{T})^+, \ \beta \in (\mathbf{N} \cup \mathbf{T})^*$$

- **Gramáticas Tipo 1**: Gramáticas sensibles al contexto o dependientes del contexto. Reconocibles con un *Autómata Linealmente Acotado* (LBA):

$$\alpha A \beta \to \alpha \nu \beta, \quad \alpha, \beta \in (\mathbf{N} \cup \mathbf{T})^*, \ \nu \in (\mathbf{N} \cup \mathbf{T})^+, \ A \in \mathbf{N}$$

- **Gramáticas Tipo 2**: Gramáticas de contexto libre (GCL) o independientes del contexto (GIC). Implementables con un *Autómata con Pila*.

$$A \to \alpha, \quad A \in \mathbf{N}, \ \alpha \in (\mathbf{N} \cup \mathbf{T})^*$$

- **Gramáticas Tipo 3**: Gramáticas regulares, implementables con un *Autómate Finito* (equivalente a una expresión regular).

$$A \to aB|a|\lambda, \quad a \in \mathbf{T}, \ A, B \in \mathbf{N}$$

1.2. Simplificación de gramáticas

Gramática limpia: No tiene reglas cadena, ni reglas-λ, ni símbolos inútiles.

Notación:

- La gramática inicial se denota $G = (\mathbf{N}, \mathbf{T}, \mathbf{P}, \mathbf{S})$ y la simplificada se denota $G = (\mathbf{N}', \mathbf{T}', \mathbf{P}', \mathbf{S}')$.
- Los nodos no terminales se denotan en letras mayúsculas, las nodos terminales en minúsculas, las letras griegas pueden denotar ambos.

1.2.1. Detección de un lenguaje vacío

1. Inicialización: $\mathcal{N} = \left\{ \forall A \in \mathbf{N} \mid (A \to t) \in \mathbf{P}, \ t \in \mathbf{T}^* \right\}$
2. Iterar hasta la convergencia:

$$\mathcal{N} = \mathcal{N} \cup \left\{ \forall B \in \mathbf{N} \mid (B \to \alpha) \in \mathbf{P}, \ \alpha \in (\mathbf{T} \cup \mathcal{N})^* \right\}$$

3. $L(G) = \emptyset \iff \mathbf{S} \notin \mathcal{N}$.

1.2.2. Eliminación de reglas λ

Símbolo anulable: $A \in \mathbf{N}$ si $\exists (A \Rightarrow^* \lambda)$.

1. Detección de símbolos anulables:
   1.1. Inicialización: $G = (\mathbf{N}, \mathbf{T}, \mathbf{P}, \mathbf{S})$ y:

$$\mathcal{N} = \left\{ \forall A \in \mathcal{N} \mid (A \to \lambda) \in \mathbf{P} \right\}$$

   1.2. Iterar hasta la convergencia:

$$\mathcal{N} = \mathcal{N} \cup \left\{ \forall B \in \mathbf{N} \mid (B \to \alpha) \in \mathcal{N}^* \right\}$$

   1.3. $\mathcal{N}$ es el onjunto de símbolos anulables.
2. $\mathbf{P}' = \mathbf{P}'$.
3. Eliminar de $\mathbf{P}'$ las reglas λ excepto $\mathbf{S} \to \lambda$ (si existe).
4. Para las reglas de la forma:

$$A \to \alpha_0 B_0 ... \alpha_k B_k, \quad k \geq 0, \ B_i \in \mathcal{N}, \ \alpha_i \notin \mathcal{N}$$

$$\mathbf{P}' = \mathbf{P}' \cup \left\{ A \to \alpha_0 X_0 ... \alpha_k X_k, \ X_i \to B_i | \lambda \right\}$$

5. Si $\mathbf{S} \in \mathcal{N}$, añadimos $\mathbf{P}' = \mathbf{P}' \cup \{\mathbf{S}' \to \mathbf{S}|\lambda\}$. Si $\mathbf{S}$ no está involucrada en la parte derecha de ninguna regla, no es necesario.

1.2.3. Eliminación de reglas unitarias

Regla unitaria: $A \to B$ tal que $A, B \in \mathbf{N}$.

Hipótesis: No hay reglas-λ.

1. Obtención de los conjuntos $\mathcal{N}_X$, $\forall X \in \mathbf{N}$.
   1.1. Inicialización: $\mathcal{N}_X = \{X\}$.
   1.2. Iterar hasta la convergencia:

$$\mathcal{N}_X = \mathcal{N}_X \cup \left\{\forall B \in \mathbf{N} \mid (A \to B) \in \mathbf{P},\ A \in \mathcal{N}_X\right\}$$

2. Si $(A \to \alpha) \in \mathbf{P}$ y no es regla unitaria, entonces $(X \to \alpha) \in \mathbf{P}$, $\forall X$ donde $A \in \mathcal{N}_X$.

1.2.4. Eliminación de símbolos inútiles

Se dice que X es un símbolo útil si:

$$S \Rightarrow^* \alpha X \beta \Rightarrow^* t, \quad t \in \mathbf{T}^*,\ \alpha, \beta \in (\mathbf{N} \cup \mathbf{T})^*$$

1. Eliminación de símbolos no terminables.

$$G' = (\mathbf{N}', \mathbf{T}', \mathbf{P}', \mathbf{S}') : \quad \forall A \in \mathbf{N}', \exists t \in \mathbf{T}^* \mid A \Rightarrow^* t$$

   1.1. Inicialización: $\mathcal{N} = \{A \mid (A \to t) \in \mathbf{P},\ t \in \mathbf{T}^*\}$.
   1.2. Iterar hasta la convergencia:

$$\mathcal{N} = \mathcal{N} \cup \left\{B \mid (B \to \alpha) \in \mathbf{P},\ \alpha \in (\mathbf{T} \cup \mathcal{N})^*\right\}$$

   1.3. Finalización: $\mathcal{N} = \mathbf{N}'$.

2. Eliminación de símbolos no accesibles:

$$G' = (\mathbf{N}', \mathbf{T}', \mathbf{P}', \mathbf{S}') : \quad \forall x \in (\mathbf{N} \cup \mathbf{T}),\ \exists \alpha, \beta \in (\mathbf{N} \cup \mathbf{T})^* \mid \mathbf{S} \Rightarrow^+ \alpha x \beta$$

   2.1. Inicialización:

$$\mathcal{N} = \{S\} \cup \{x \mid (S \to \alpha x \beta) \in \mathbf{P},\ x \in (\mathbf{N} \cup \mathbf{T}),\ \alpha, \beta \in (\mathbf{N} \cup \mathbf{T})^*\}$$

   2.2. Iterar hasta la convergencia:

$$\mathcal{N} = \mathcal{N} \cup \left\{y \mid (A \to \alpha y \beta) \in \mathcal{N},\ y \in (\mathbf{N} \cup \mathbf{T}),\ \alpha, \beta \in (\mathbf{N} \cup \mathbf{T})^*\right\}$$

   2.3. Finalización:

$$\mathbf{N}' = \mathcal{N} \cap \mathbf{N}, \qquad \mathbf{T}' = \mathcal{N} \cap \mathbf{T}$$

$$\mathbf{P}' = \left\{(A \to \alpha) \mid A \in \mathbf{N}'^*,\ \alpha \in (\mathbf{N}' \cup \mathbf{T}')^*\right\}$$

1.3. Análisis Léxico (*scanners*)

Un autómata finito (AF) se deine como la tupla:

$$\mathrm{AF} = (\mathrm{Q}, \mathrm{Te}, \delta, q_0, \mathrm{F})$$

- Q: Conjunto de estados.
- Te: Alfabeto de entrada.
- q_0: Estado inicial.
- F $\subset Q$: Conjunto de estados finales.
- δ: Función $\mathrm{Q} \times \{\mathrm{Te} \cup \{\lambda\}\} \to P(\mathrm{Q})$.

1.3.1. Conversión de una gramática regular a un Autómata Finito

Sea una gramática regular de la forma:

$$G = (\mathbf{N}, \mathbf{T}, \mathbf{P}, \mathbf{S}), \qquad A \to aB,\ A \to a,\ A \to \lambda, \quad a \in \mathbf{T},\ A, B \in \mathbf{N}$$

- $Q = \mathbf{N} \cup \{Z\}$, $Z \notin \mathbf{N}$ si $\exists (A \to a) \in \mathbf{P}$. En otro caso $Q = \mathbf{N}$.
- Añadimos las transiciones de estados:
  - $\delta(A, a) = B$ si $(A \to aB) \in \mathbf{P}$.
  - $\delta(A, a) = Z$, si $(A \to a) \in \mathbf{P}$.
- Añadimos los estados finales:
  - Si $\mathbf{Z} \in$ Q: $F = \{A \mid (A \to \lambda) \in \mathbf{P}\} \cup Z$.
  - En otro caso, $F = \{A \mid (A \to \lambda) \in \mathbf{P}\}$.

1.3.2. Algoritmo de Thomson

Permite transformar una expresión regular en un Autómata Finito No Determinista con transiciones λ (AFND-λ).

Tema 2: Modelos de Lenguaje

2.1. Smoothing

- **Laplace smoothing (add-one)**:

$$\mathbb{P}_{\text{Laplace}}(x) = \frac{C(x)+1}{N+V}$$

where V is the size of the lexicon of tokens and N is the total number of tokens in the corpus.

- **Good-Turing smoothing**: Replace count c by the smoothing count c^*:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

  - N_c: Number of n-grams with count c.
  - N_0: approached by means of N.
  - c^* is only used for small values of c.
  - Approximations when $N_{c+1} = 0$.

- **Interpolation**:

$$\mathbb{P}(w_i|w_{i-1}, w_{i-2}) = \lambda_1\mathbb{P}(w_{i-1}|w_i, w_{i-2}) + \lambda_2\mathbb{P}(w_i|w_{i-2}) + \lambda_3\mathbb{P}(w_i)$$

given $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

- **Backoff**:

$$\mathbb{P}_{\text{katz}}(w_i|w_{i-1}, w_{i-2}) = \begin{cases} \mathbb{P}^*(w_i|w_{i-1}w_{i-2}) & C(w_{i:i-2}) > 0 \\ \alpha(w_{i-1}w_{i-2})\mathbb{P}^*(w_i|w_{i-1}) & C(w_{i-1:i-2}) > 0 \\ \alpha(w_{i-1})\mathbb{P}^*(w_{i-2}) & \text{otherwise} \end{cases}$$

where $\mathbb{P}^*$ are smoothed probabilities à la *Good-Turing* and α mus guarantee that the total probability mass sums up 1.

- **Absolute discounting**:

$$\mathbb{P}_{\text{AbsDisc}}(w_i|w_{i-1}) = \frac{C(w_{i-1}w_i) - d}{\sum_\nu C(w_{i-1}\nu)} + \lambda(w_{i-1})\mathbb{P}(w_i)$$

- **Kneser-Ney Smoothing**:

$$\mathbb{P}_{\text{KN}}(w_i|w_{i-n+1:i-1}) = \frac{\text{máx}\{c_{\text{KN}}(w_{i-n+1:i}) - d, 0\}}{\sum_\nu c_{\text{KN}}(w_{i-n+1:i-1}\nu)} + \lambda(w_{i-n+1:i-1})\mathbb{P}_{\text{KN}}(w_i|w_{i-n+2:i-1})$$

$$c_{\text{KN}}(\cdot) = \begin{cases} \text{count}(\cdot) & \text{highest order} \\ \text{continuationcount}(\cdot) & \text{lower orders} \end{cases}, \qquad \lambda(\cdot) = \frac{d}{\sum_\nu C(\cdot\nu)}|\{w : C(\cdot w) > 0\}|$$

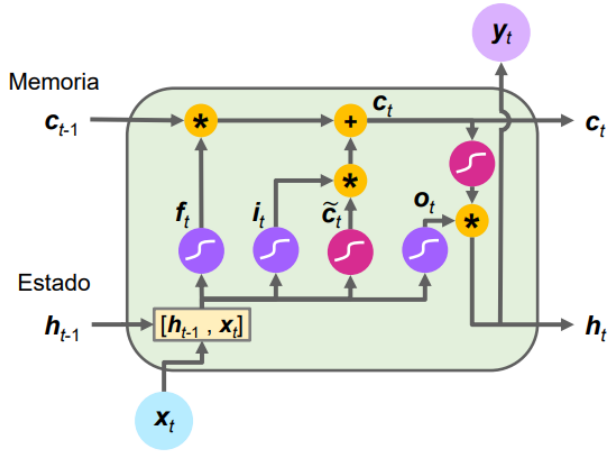
where the *continuation count* is the number of unique single word contexts for $\cdot$.

At the termination of the recursion:

$$\mathbb{P}_{\text{KN}} = \frac{\text{máx}\{c_{\text{KN}}(w) - d, 0\}}{\sum_{w'} c_{\text{KN}}(w')} + \lambda(\varepsilon)\frac{1}{V}$$

2.2. LSTM (Long-Short Term Memory)

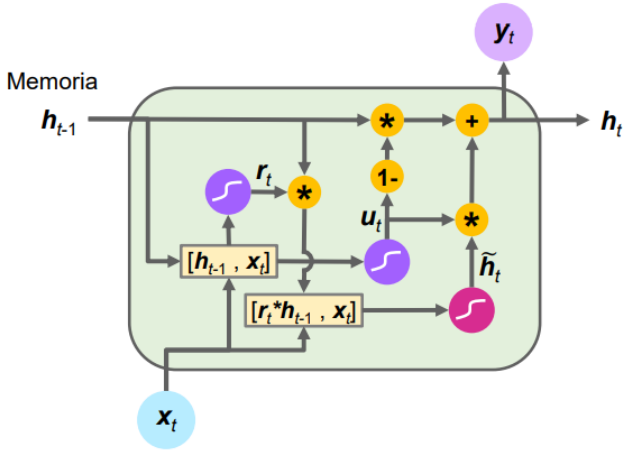
Figura 2.1: LSTM structure



1. Forget gate: $\mathbf{f}_t = \text{sig}(\mathbf{W_f}[\mathbf{h}_{t-1}\mathbf{x}_t] + \mathbf{b_f})$.
2. Input gate: $\mathbf{i}_t = \text{sig}(\mathbf{W_i}[\mathbf{h}_{t-1}\mathbf{x}_t] + \mathbf{b_i})$.
3. New candidate: $\tilde{\mathbf{c}}_t = \tanh(\mathbf{W_c}[\mathbf{h}_{t-1}\mathbf{x}_t] + \mathbf{b_c})$
4. New memory: $\mathbf{c}_t = \mathbf{f}_t\mathbf{c}_t + \imath_t\tilde{\mathbf{c}}_t$.
5. Output gate: $\mathbf{o}_t = \text{sig}(\mathbf{W_o}[\mathbf{h}_{t-1}\mathbf{x}_t] + \mathbf{b_o})$
6. New state: $\mathbf{h}_t = \mathbf{o}_t * \tanh(\mathbf{c}_t)$

2.3. GRU (Gated Recurrent Unit)

Figura 2.2: GRU structure



1. Reset gate: $\mathbf{r}_t = \text{sig}(\mathbf{W_r}[\mathbf{h}_{t-1}\mathbf{x}_t] + \mathbf{b_r})$
2. New candidate: $\tilde{\mathbf{h}}_t = \tanh(\mathbf{W_h}[\mathbf{r}_t * \mathbf{h}_{t-1}\mathbf{x}_t] + \mathbf{b_h})$
3. Update gate: $\mathbf{u}_t = \text{sig}(\mathbf{W_u}[\mathbf{h}_{t-1}\mathbf{x}_t] + \mathbf{b_u})$.
4. New state: $\mathbf{h}_t = \mathbf{u}_t * \tilde{\mathbf{h}}_t + (1 - \mathbf{u}_t)\mathbf{h}_{t-1}$

Tema 3: Análisis morfológico del lenguaje natural

- **Parts of speech** (POS, *"categorías gramaticales"*): noun, verb, pronoun, preposition, adverb, conjucion, participle and article.
- **Named entity**: Anything that can be referred to with a proper name.
- Parts of speech and named entities are useful clues to sentence structure and meaning.
- **Part-of-speech taggin**: Taking a sequence of words and assigning each word a part of speech.
- **Named entity recognition** (NER): Assigning words or phrases tags like PERSON, LOCATION or ORGANIZATION.
- **Sequence labeling tasks**: Tasks where each word x_i in an input word sequence is assigned a label y_i, so the output sequence Y has the same length as the input sequence X.
- Hidden Markov Model (HMM): generative sequence labeling algorithm.
- Conditional Random Field (CRF): discriminative sequence labeling algorithm.

3.1. Part-of-speech Tagging

- Tagging is a disambiguation task: words are ambiguous. The goal of POS tagging is to resolve these ambiguities, choosing the proper tag for the context.
- The accuracy of POS tagging algorithms is extremely high.
- Most word types (85-85 %) are unambiguous, but the ambiguous words are very common.
- Many words are easy to disambiguate because their different tags are not equally likely. Thus, given an ambiguous word, choose the tag which is mos frequent in the training corpus.

3.2. Named Entity Tagging

Goal: Find spans of text that constitute proper names and tag the type of the entity.

- Common entity tags: PER (person), LOC (location), ORG (organization) or GPE (geo-political entity).
- Task: find and label spans of text (where the boundaries are).

BIO tagging is the standard approach for NER. This method allows to treat NEW like a word-by-word sequence labeling task via tags that capture the boundary and the named entity type.

- Any token that beings a span of interest is tagged with the label B.
- Tokens that occur inside a span are tagged with an I.
- Tokens outside of any span of interest are labeled O.
- Alternatives: IO tagging, BIOES tagging.

3.3. HMM for POS tagging

An HMM is a probabilistic sequence model: given a sequence of units, it computes a probability distribution over possible sequences of labels and chooses the best label sequence.

3.3.1. Markov Chain

Assumption: To predict the future in the sequence, all that matters is the current state.

Consider a sequence of state variables $q_1, ...q_i$:

$$\mathbb{P}(q_i = a|q_1, ..., q_{i-1}) = \mathbf{P}(q_i = a|q_{i-1})$$

Components of Markov chain:

- $Q = q_1, ..., q_N$: Set of N states.
- $A = \left(a_{ij}\right),\ i, j \in \{1, ..., N\}^2$: Transition probability matrix where a_{ij} is the probability of moving from state i to state j.
- $\pi = \pi_1, ..., \pi_N$: Initial probability distribution over states, i.e. π_i is the probability that the Markov chain will start in state i.

3.3.2. The Hidden Markov Model

Key: Events of interest are hidden so they are not directly observable.

A Hidden Markov Model (HMM) allows to talk about *observed* events (words) and *hidden* events (POS tags) that are thought as causal factors in the probabilistic model.

Componentes of HMMs:

- $Q = q_1, ..., q_N$: Set of N states.
- $A = \left(a_{ij}\right),\ i, j \in \{1, ..., N\}^2$: Transition probability matrix where a_{ij} is the probability of moving from state i to state j.
- $O = o_1, ..., o_T$: Sequence of T observations drawn from a vocabulary $V = v_1, ..., v_V$.
- $B = b_i(o_t)$: Sequence of observation likelihoods (**emision probabilities**) expressing the probability of an observation o_t being generated from state q_i.
- $\pi = \pi_1, ..., \pi_N$: Initial probability distribution over states, i.e. π_i is the probability that the Markov chain will start in state i.

Assumption: The probability of an output observation o_i depends only on the state that produced the observation q_i:

$$\mathbb{P}(o_i|q_1, ..., q_i, ..., q_T, o_1, ..., o_i, ..., o_T) = \mathbb{P}(o_i|q_i)$$

The A matriz contains the tag transition probabilities, which represent the probability of a tag occurring given the previous tag. The maximum likelihood estimate is:

$$\mathbb{P}(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$

The B emission probabilities, $\mathbb{P}(w_i|t_i)$ represent the probability, given a tag, that it will be associate with a given word. The ML of the emission probability is:

$$\mathbb{P}(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$

3.3.3. HMM tagging as decoding

Decoding: Given as input an HMM $\lambda = (A, B)$ and a sequence of observations $O = o_1, ..., o_T$, find the most probable sequence of states $Q = q_1, ..., q_T$.

The goal of HMM decoding is to choose the tag sequence $t_1, ..., t_n$ that is mot probable given the observation sequence of n words $w_1, ..., w_n$:

$$\begin{aligned} \hat{t}_{1:n} &= \underset{t_1,...,t_n}{\arg\max} \mathbb{P}(t_1, ..., t_n|w_1, ..., w_n) \\ &\underset{\text{Bayes rule}}{=} \underset{t_1,...,t_n}{\arg\max} \frac{\mathbb{P}(w_1, ..., w_n|t_1, ..., t_n)\mathbb{P}(t_1, ..., t_n)}{\underbrace{\mathbb{P}(w_1, ..., w_n)}_{\text{ignore}}} \end{aligned} \tag{3.1}$$

- Assumption: probability of a word appearing depends only on its own tag:

$$\mathbb{P}(w_1, ..., w_n|t_1, ..., t_n) \approx \prod_{i=1}^{n} \mathbb{P}(w_i|t_i)$$

- Assumption: Probability of a tag is dependen only on the previous tag.

$$\mathbb{P}(t_1, ..., t_n) \approx \prod_{i=1}^{n} \mathbb{P}(t_i|t_{i-1})$$

Reformulate the equation 3.1:

$$\hat{t}_{1:n} = \arg\max_{t_1,...,t_n} \prod_{i=1}^{n} \overbrace{\mathbb{P}(w_i|t_i)}^{\text{emission}} \underbrace{\mathbb{P}(t_i|t_{i-1})}_{\text{transition}}$$

3.3.4. The Viterbi Algorithm

Goal: Find the optimal sequence of tags.

The Viterbi algorithm first sets up a probability matrix (**lattice**) with one column for observation o_t and one row for state q_j. Thus, each cell $v_t(j)$ represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable sequence $q_1, ..., q_{t-1}$ given the HMM λ.

$$v_t(j) = \max_{q_1,...,q_{t-1}} \mathbb{P}(q_1, ..., q_{t-1}, o_1, ..., o_t, q_t = j|\lambda)$$

Viterbi fills each cell recursively:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t)$$

- $v_{t-1}(i)$: the **previous Viterbi path probability**.
- a_{ij}: the **transition probability** from previous state q_i to current state q_j.
- $b_j(o_t)$: the **state observation likelihood** of o_t given the current state j.

Note: If $t = 1$, $v_1(j)$ is filled:

$$v_1(j) = \pi_j b_j(o_1)$$

Problem: HMM needs a number of augmentations to achieve high accuracy. It would be great to have ways to add arbitrary features based on capitalization or morphology. In general it's hard for generative models like HMMs to add these features directly in the model in a clean way

3.4. Conditional Random Fields (CRFs)

In a CRF, the output sequence is computed directly via the posterior probability:

$$\hat{Y} = \arg\max_{Y \in \mathcal{Y}} \mathbb{P}(Y|X)$$

A CRF is a log-lineal model that assigns a probability to an entire output sequence Y out of all possible sequences $\mathcal{Y}$ given the entire input sequence X.

The function F maps an entire input sequence X and a entire output sequence Y to a feature vector. Assume K features with weight w_k for each feature F_k (called **global features**).

$$\mathbb{P}(Y|X) = \frac{1}{Z(X)} \exp\Big(\sum_{k=1}^{K} w_k F_k(X, Y)\Big)$$

$$Z(X) = \sum_{Y' \in \mathcal{Y}} \exp\Big(\sum_{k=1}^{K} w_k F_k(X, Y')\Big)$$

Each feature is computed by decomposing it into a sum of **local features** for each position i in Y:

$$F_k(X, Y) = \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i)$$

where each f_k (local feature) is allowed to make use of:

- y_i: current output token.
- y_{i-1}: previous output token.
- X: entire input string.
- i: current position.

In a CRF we don't learn weights for each of these local features f_k. Instead, we first sum the values of each local feature over the entire sentence to create each global feature F_k. It is those global features that are multiplied by weight w_k. Thus, for training and inference there is always a fixed set of K features with K weights, even though the length of each sentence is different.

$$\begin{aligned}
\hat{Y} &= \arg\max_{Y\in\mathcal{Y}} \mathbb{P}(Y|X) \\
&= \arg\max_{Y\in\mathcal{Y}} \underbrace{\frac{1}{Z(X)}}_{\text{ignore}} \exp\left(\sum_{k=1}^{K} w_k F_k(X,Y)\right) \\
&= \arg\max_{Y\in\mathcal{Y}} \exp\left(\sum_{k=1}^{K} w_k \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i)\right) \\
&= \arg\max_{Y\in\mathcal{Y}} \sum_{k=1}^{K} w_k \sum_{i=1}^{n} f_k(y_{i-1}, y_i, X, i) \\
&= \arg\max_{Y\in\mathcal{Y}} \sum_{i=1}^{n} \sum_{k=1}^{K} w_k f_k(y_{i-1}, y_i, X, i)
\end{aligned}$$

Conclusion: Viterbi algorithm to decode the optimal tag sequence $\hat{Y}$.

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) \sum_{k=1}^{K} w_k f_k(y_{t-1}, y_t, X, t), \qquad j = 1, ..., N,\ t = 2, ..., T$$

Tema 4: Hidden Markov Models

Markov models should be characterized by three fundamental problems:

1. **Likelihood**: Given an HMM $\lambda = (A, B)$ and an observation sequence O, determine the likelihood $\mathbb{P}(O|\lambda)$.

2. **Decoding**: Given an observation sequence O and an HMM $\lambda = (A, B)$, discover the best hidden state sequence Q.

3. **Learing**: Given an observation sequence O and the set of states in the HMM, learn the HMM parameters A and B.

4.1. Likelihood Computation. The Forward Algorithm

Computing likelihood. Given an HMM $\lambda = (A, B)$ and a observation sequence O, determine the likelihood $\mathbb{P}(O\lambda)$.

Problem: The hidden states are not observables.

Assume we already know the hidden states. The likelihood of the observation sequence given the particular hidden state sequence is:

$$\mathbb{P}(O|Q) = \prod_{i=1}^{T} \mathbb{P}(o_i|q_i)$$

Since we don't know the sequence Q, let's compte the joint probability of being in a particular sequence Q and generating a particular sequence O:

$$\mathbb{P}(O, Q) = \mathbb{P}(O|Q) \cdot \mathbb{P}(Q) = \prod_{i=1}^{T} \mathbb{P}(o_i|q_i) \prod_{i=1}^{T} \mathbb{P}(q_i|q_{i-1})$$

And we can compute the **total probability of the observations** by summing over all possible hidden state sequences:

$$\mathbb{P}(O) = \sum_{Q} \mathbb{P}(O, Q) = \sum_{Q} \mathbb{P}(O|Q)\mathbb{P}(Q)$$

- Problem: For and HMM with N hidden states and a observation sequence of T observations there are N^T possible hidden sequences.
- Solution: Use an efficient $O(N^2T)$ algorithm called the **forward algorithm** based on dynamic programming. This algorithm computes the osbervation probability by summing over the probabilities of all possible hidden state paths

Let's assume we have a table with $N \times T$ dimensiones (where N is the number of hidden states and T is the length of the observation sequence). Each cell $\alpha_t(j)$ represents the probability of being in state j after seeing the first t observations given λ, thus, it is computed by summing over the probabilities of every path that could lead us to this cell.

$$\begin{aligned}\alpha_t(j) &= \mathbb{P}(o_1, ..., o_t, q_t = j|\lambda) \\ &= \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t)\end{aligned}$$

- $\alpha_{t-1}(i)$: the **previous forward path probability** from previous time step.
- a_{ij}: the **transition probability** from previous state q_i to current state q_j.

- $b_j(o_t)$: the **state observation likelihood** of the observation symbol o_t given the current state j.

1. Initialization:
$$\alpha_1(j) = \pi_j b_j(o_1), \quad j = 1, ..., N$$
2. Recursion:
$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t), \quad j = 1, ..., N, \ t = 2, ..., T$$
3. Termination:
$$\mathbb{P}(O|\lambda) = \sum_{i=1} \alpha_T(i)$$

4.2. Decoding: The Viterbi Algorithm

Decoding: Given as input an HMM $\lambda = (A, B)$ and a sequence of observation $O = o_1, ..., o_T$, find the most probable sequence of states $Q = q_1, ..., q_T$.

The Viterbi algorithm (based on dynamic programming) process the observation sequence filling a table of $N \times T$ dimensions. Each cell $v_t(j)$ represents the probability that the HMM is instate j after seeing the first t observations and passing through the most probable state sequence $q_1, ..., q_{t-1}$.

$$v_t(j) = \max_{q_1,...,q_{t-1}} \mathbb{P}(q_1, ..., q_{t-1}, o_1, ..., o_t, q_t = j|\lambda)$$
$$= \max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t)$$

- $v_{t-1}(i)$: the **previous Viterbi path probability** from the previous time step.
- a_{ij}: the **transition probability** from previous state q_i to current state q_j.
- $b_j(o_t)$: the **state observation likelihood** of the observation symbol o_t given the current state j.

Viterbi algorithm has **backpointers**, since it must produce a probability and the most likely state sequence.

1. Initialization:
$$\begin{aligned} v-1(j) &= \pi_j b_j(o_1) \quad j = 1, ..., N \\ bt_1(j) &= 0 \quad j = 1, ..., N \end{aligned}$$
2. Recursion:
$$\begin{aligned} v_t(j) &= \text{máx}_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t) \quad j = 1, ..., N, \ t = 2, ..., T \\ bt_t(j) &= \arg\text{máx}_{i=1,...,N} N v_{t-1}(i) a_{ij} b_j(o_t) \quad j = 1, ..., N, \ t = 2, ..., T \end{aligned}$$
3. Termination:
$$\begin{aligned} \text{Best score:} \quad P^* &= \text{máx}_{i=1}^{N} v_T(i) \\ \text{Start of backtrace:} \quad q_T^* &= \arg\text{máx}_{i=1,...,N} v_t(i) \end{aligned}$$

4.3. HMM Training: The Forward-Backward Algorithm (Baum-Welch)

Learning: Given an observation sequence O and the set of possible states in the HMM, learn the HMM parameters A and B.

The forward-backward algorithm is a special case of the Expectation-Maximization algorithm. It train both the transition probabilities A and emission probabilities B of the

HMM. It is an iterative algorithm, computing an initial estimate for the probabilities, then using those estimates to compute a better estimate, and so on, iteratively improving the probabilities that it learns.

Le's define the **backward probability** β as the probability of seeing the observations from time $t+1$ to the end, given that we are in state i at time t (and given the automaton λ):

$$\beta_t(i) = \mathbb{P}(o_{t+1}, ..., o_T | q_t = i, \lambda)$$

And it is computed inductively in a similar manner to the forward algorithm.

1. Initialization:
$$\beta_T(i) = 1$$
2. Recursion:
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij}\beta_j(o_{t+1})\beta_{t+1}(j), \quad i = 1, ..., N, \ t = 1, ..., T-1$$
3. Termination:
$$\mathbb{P}(O|\lambda) = \sum_{j=1}^{N} \pi_j b_j(o_1)\beta_1(j)$$

We can estimate $\hat{a}_{ij}$ by a variant of simple maximum likelihood estimation:

$$\hat{a}_{ij} = \frac{\text{expected number of transitions from state } i \text{ to state } j}{\text{expected number of transitions from state } i}$$

Assumption: We have some estimate of the probability that a given transition $i \to j$ was taken at a particular point in time t in the observation sequence O. If we know this probability for each t, we can sum over all times t to estimate the total count for the transition $i \to j$.

Let ξ_t be the probability of being in state i at time t and state j at time $t+1$ given the observation sequence and the model:

$$\xi_t(i,j) = \mathbb{P}(q_t = i, q_{t+1} = j | O, \lambda)$$

Let not-quite-$\xi_t(i,j)$:

$$\begin{aligned}\text{not-quite-}\xi_t(i,j) &= \mathbb{P}(q_t = i, q_{t+1} = j, O|\lambda) \\ &= \alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)\end{aligned}$$

So the final equation for ξ_t is:

$$\begin{aligned}\xi_t(i,j) &= \frac{\text{not-quite-}\xi_t(i,j)}{\mathbb{P}(O|\lambda)} \\ &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)}\end{aligned}$$

The expected number of transitions from state i to state j is then the sum over all t of ξ.

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1}\xi_t(i,j)}{\sum_{t=1}^{T-1}\sum_{k=1}^{N}\xi_t(i,k)}$$

We can estimate $\hat{b}_j(v_k)$, the probability of given symbol v_k from the observation vocabulary V given a state j:

$$\hat{b}_j(v_k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j}$$

Let $\gamma_t(j)$ be th probability of being in state j at time t:

$$\gamma_t(j) = \mathbb{P}(q_t = j|O, \lambda) = \frac{\mathbb{P}(q_t = j, O|\lambda)}{\mathbb{P}(O|\lambda)} = \frac{\alpha_t(j)\beta_t(j)}{\mathbb{P}(O|\lambda)}$$

So we can compute b:

$$\hat{b}_j(v_k) = \frac{\sum_{t=1, o_t=v_k} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}$$

The forward-backward algorithm starts with some initial estimate for $\lambda = (A, B)$ and iteratively runs two steps:

1. E-step (expectation): Compute the expected state occupancy count γ and the expected state transition count ξ from earlier A and B probabilities.

$$\begin{aligned} \gamma_t(j) &= \frac{\alpha_t(j)\beta_t(j)}{\alpha_T(q_F)} && \forall t, j \\ \xi_t(i,j) &= \frac{\alpha_t(i)a_{ij}b_j(o_{t+1})\beta_{t+1}(j)}{\alpha_T(q_F)} && \forall t, i, j \end{aligned}$$

2. M-step (maximization): Use γ and ξ to recompute new A and B probabilities.

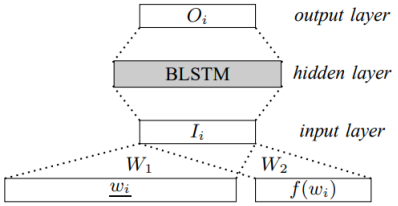
$$\begin{aligned} \hat{a}_{ij} &= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{k=1}^{N} \xi_t(i,k)} \\ \hat{b}_j(v_k) &= \frac{\sum_{t=1, o_t=v_k} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \end{aligned}$$

Tema 5: Neural Models for PoS tagging

5.1. General bidirectional LSTM [Wang et al., 2015]

Given a $w = w_1, ..., w_n$ with tags $y_1, ..., y_n$, Bi-LSTM is used to predict the tag probability distribution of each word.

Figura 5.1: Bi-LSTM schema

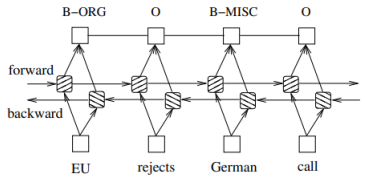


- w_i: one-hot representation of the word in position i in the sentence.
- $f(w_i)$: three-dimensional binary vector to tell if w_i is full lowercase, full uppercase or leading with a capital letter.
- $I_i = W_1 w_i + W_2 f(w_i)$.
- $W_1 w_i$: word embedding of w_i (since W_1 is the embeddings matrix).
- For words without corresponding external embeddings, their word embeddings are initialized with uniformly distributed random values from -0.1 to 0.1.
- The class of word-embeddings does not seem to be relevant in results.

5.2. Bidirectional LSTM-CRF [Huang et al., 2015]

- Bidirectional LSTM with a CRF layer (BI-LSTM-CRF) efficiently uses past and future input features thanks to bidirectional LSTM componente.
- It also uses sentence level tag information thanks to a CRF layer.
- Conv-CRF: Model which consits of a convolutional network and CRF layer on the output. It has generated promising results on sequence tagging tasks.

Figura 5.2: Bi-LSTM-CRF schema



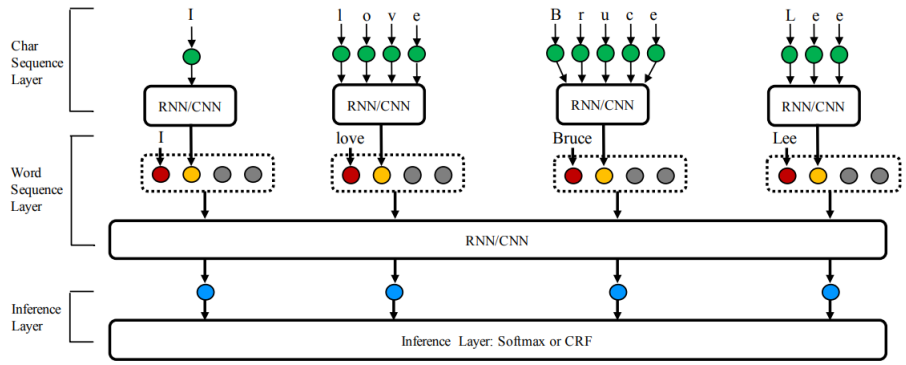
5.3. NCRF++ Architecture [Yang and Zhang, 2018]

- Layer 1: character sequence layer.
- Layer 2: word sequence layer.
- Layer 3: inference layer.

5.3.1. Character Sequence Layer

Input: Character sequence information (per word) represented with **character embeddings** (green circles).

Figura 5.3: NCRF++ schema



- **Character RNN** to capture the left-to-right and right-to-left sequence information and concatenates the finall hidden states of two RNNs as the encoder of the input character sequence.
- **Character CNN**: takes a sliding window to capture local features and then uses max-pooling for aggregated encoding of the character sequence.

5.3.2. Word Sequence Layer

Input: Word embeddings (red circles), character sequence encoding hidden vector (yellow circles) and handcrafted neural features (gray circles).

- **Word RNN**: Bidirectional RNNs are supported to capture left and right contexted information of each word. The hidden vectors for both directions on each word are concatenated to represent the corresponding word.
- **Word CNN**: Utilizes the same sliding window as character CNN while nonlinear function is attached with extracted features.

5.3.3. Inference Layer

- Input: Extracted word sequence representations as features.
- Output: Labels to the word sequence.
- A linear layer firstly maps the input to label vocabulary size scores, which are used to:
  a) Model the label probabilities of each word through simple softmax.
  b) Caculate the label score of the whole sentence.
- **Softmax**: Maps the label scores into probability space.
- **CRF**: captures label dependencies by adding transition scores between neighboring labels. During decoding process, the Viterbi algorithm is used to search the label sequence with the highest probability.

Bibliografía

[Huang et al., 2015] Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

[Wang et al., 2015] Wang, P., Qian, Y., Soong, F. K., He, L., and Zhao, H. (2015). Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. *arXiv preprint arXiv:1510.06168*.

[Yang and Zhang, 2018] Yang, J. and Zhang, Y. (2018). Ncrf++: An open-source neural sequence labeling toolkit. *arXiv preprint arXiv:1806.05626*.