IIC3670 Procesamiento de Lenguaje Natural

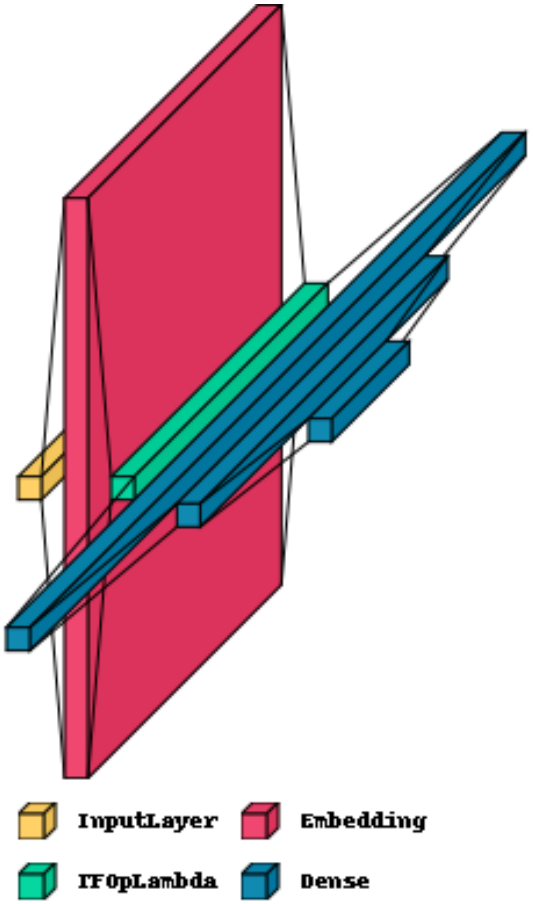
https://github.com/marcelomendoza/IIC3670

Clasificación con word embeddings

| V |

```
inputs = Input(shape=(max_tokens, ))

embeddings_layer = Embedding(input_dim=len(tokenizer.index_word)+1,
output_dim=embed_len, input_length=max_tokens, trainable=False,
weights=[glove_50_embeddings])

dense1 = Dense(128, activation="relu")
dense2 = Dense(64, activation="relu")
dense3 = Dense(len(classes), activation="softmax")

x = embeddings_layer(inputs)
x = tensorflow.reduce_sum(x, axis=1)
x = dense1(x)
x = dense2(x)
outputs = dense3(x)

model = Model(inputs=inputs, outputs=outputs)
```

forward



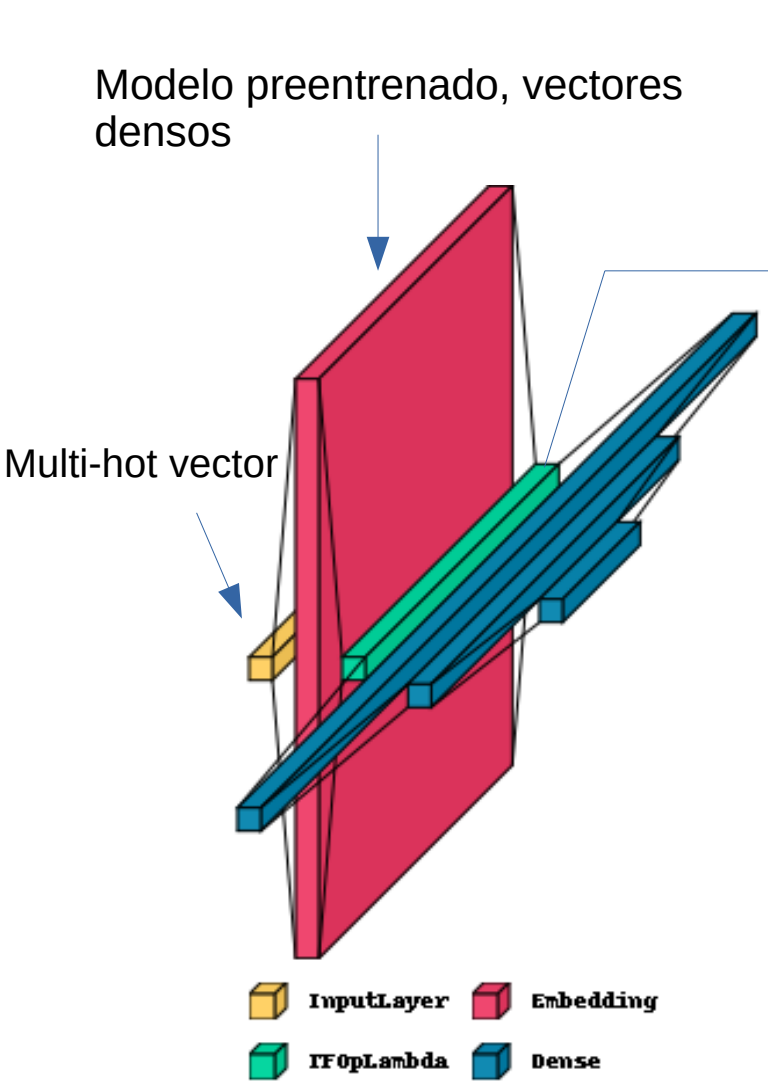
Clasificación con word embeddings

Modelo preentrenado, vectores densos

Word embeddings

tensorflow.reduce_sum(x, axis=1)

Multi-hot vector



- SUBWORDS, FASTTEXT -

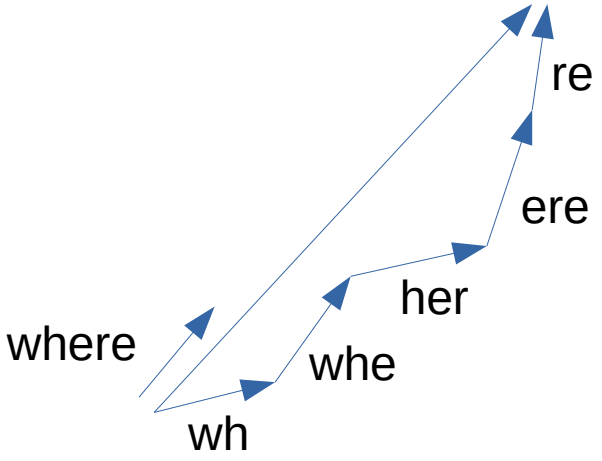
Subwords

Las palabras se representan por un **char *n*-grams**:

Ej.: *n* = 3, where → <wh, whe, her, ere, re>, <where>

Sea $\mathcal{G}_w \subset \{1, \ldots, G\}$ el conjunto de char *n*-grams de la palabra *w.* En FastText, cada *n*-gram tiene un vector $\mathbf{z}_g$ que lo representa.

Una palabra se representa como la **suma** de los vectores $\mathbf{z}_g$, en $\mathcal{G}_w \subset \{1, \ldots, G\}$



Subwords

FastText es una extensión de skip-grams basada en sub-words.

La función objetivo de skipgrams corresponde a la log verosimilitud:

$$\sum_{t=1}^{T} \sum_{c \in \mathcal{C}_t} \log p(w_c \mid w_t),$$

Podemos usar una softmax para definir la probabilidad de una palabra de contexto:

$$p(w_c \mid w_t) = \frac{e^{s(w_t,\, w_c)}}{\sum_{j=1}^{W} e^{s(w_t,\, j)}}.$$

Las palabras de contexto son ejemplos positivos. Usamos negative sampling para que el problema sea de clasificación binaria.

Bojanovski et al. (2016) Enriching Word Vectors with Subword Information, https://arxiv.org/pdf/1607.04606.pdf

Subwords

Cuando consideramos *negative sampling*, la función objetivo de clasificación binaria corresponde a una *log* loss:

$$\log\left(1 + e^{-s(w_t,\, w_c)}\right) + \sum_{n \in \mathcal{N}_{t,c}} \log\left(1 + e^{s(w_t,\, n)}\right)$$

↳ Negative samples (fuera del contexto de w_t)

Subwords

Cuando consideramos *negative sampling*, la función objetivo de clasificación binaria corresponde a una *log* loss:

$$\log\left(1 + e^{-s(w_t,\, w_c)}\right) + \sum_{n \in \mathcal{N}_{t,c}} \log\left(1 + e^{s(w_t,\, n)}\right)$$

↳ Negative samples (fuera del contexto de w_t)

Luego, tenemos varios problemas de clasificación binaria independientes que superponemos durante el entrenamiento:

$$\sum_{t=1}^{T} \left[\sum_{c \in \mathcal{C}_t} \ell(s(w_t,\, w_c)) + \sum_{n \in \mathcal{N}_{t,c}} \ell(-s(w_t,\, n)) \right]$$

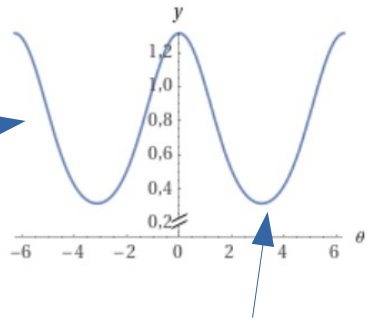
donde: $\ell : x \mapsto \log(1 + e^{-x})$

Subwords

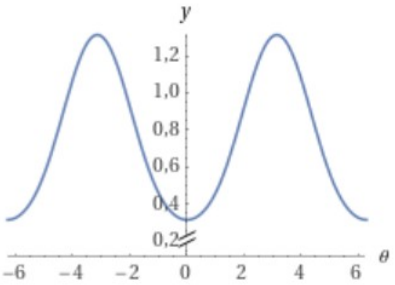
x → ~ 0 → e^{0} → ~ 0 (mínima pérdida)

Log loss:

$$\log\left(1+e^{-s(w_t,\,w_c)}\right)+\sum_{n\in\mathcal{N}_{t,c}}\log\left(1+e^{s(w_t,\,n)}\right)$$



pi/2



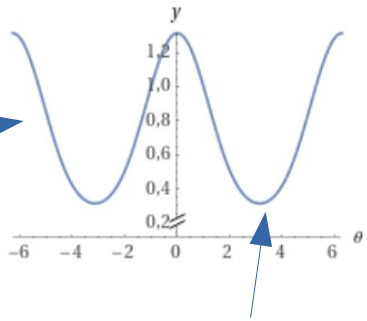
x → ~ 1 → e^{-0} → ~ 0 (mínima pérdida)

Subwords

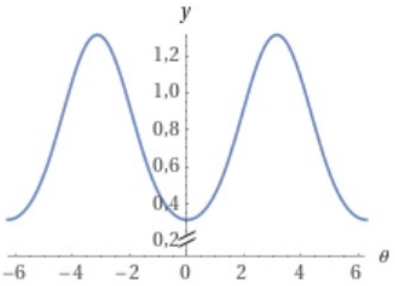
x → ~ 0 → e^{0} → ~ 0 (mínima pérdida)

Log loss:

$$\log\left(1+e^{-s(w_t,\,w_c)}\right)+\sum_{n\in\mathcal{N}_{t,c}}\log\left(1+e^{s(w_t,\,n)}\right)$$



pi/2



x → ~ 1 → e^{-0} → ~ 0 (mínima pérdida)

FastText se entrena usando como función de scoring:

$$s(w,c)=\sum_{g\in\mathcal{G}_w}\mathbf{z}_g^{\top}\mathbf{v}_c.$$

Entrenamiento de FastText

- Se entrenó en base a la implementación en C de skipgram y CBOW del word2vec package de Google.

- Optimización: SGD con linear decay, en paralelo usando Hogwild.

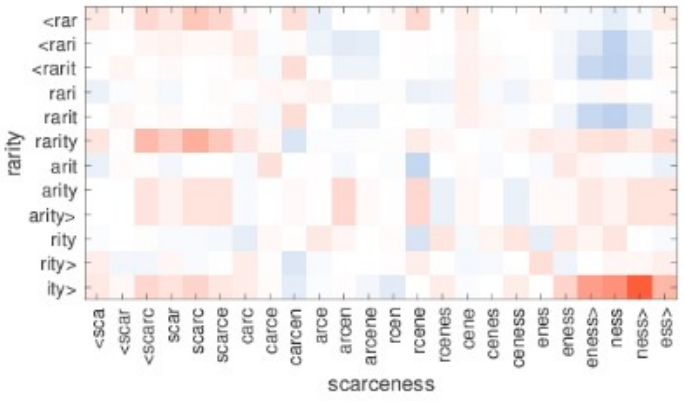
- Dims de embeddings: 300; Por cada + ex. 5 – exs at random (según curva de frecuencia); ventana de contexto = 5.

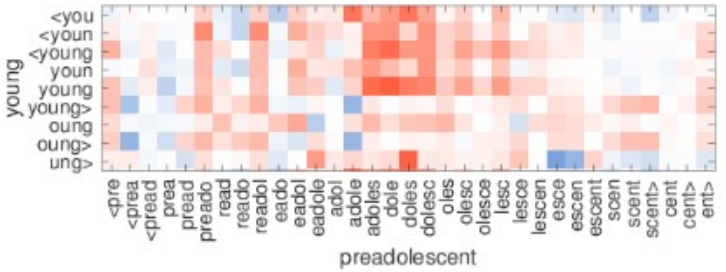
- Implementación del modelo en https://github.com/facebookresearch/fastText

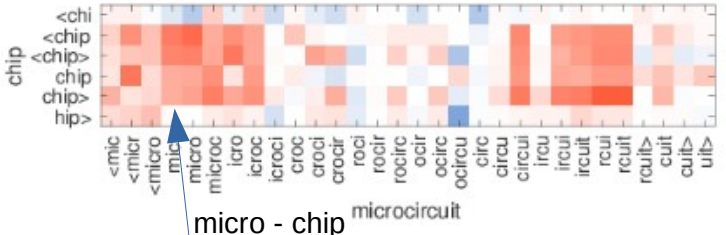
- Data provenance: Wikipedia dumps (https://dumps.wikimedia.org/), original en 9 idiomas (incluído el español), versión posterior con ~140 idiomas.

Subwords

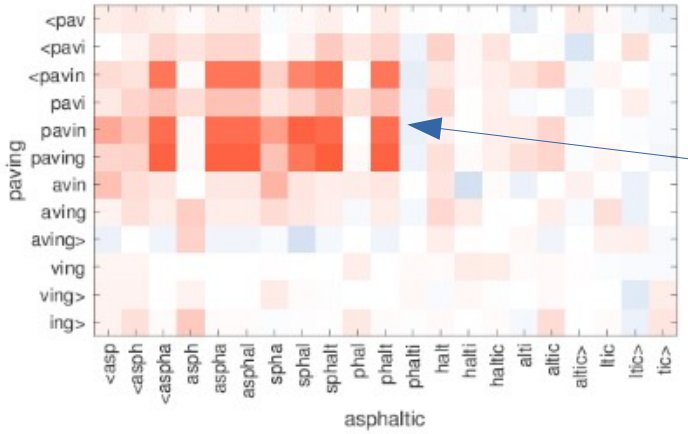
- FastText captura palabras out of vocabulary y palabras poco frecuentes
- En algunos idiomas obtiene resultados mucho mejores que word2vec







micro - chip



Hay algo de semántica en la correlación de char *n*-grams

Aspectos prácticos:

- FastText mapea los *n*-grams a enteros usando hashing.
- App. 2 millones de char *n*-grams se usan en FastText.

Subwords

	word	n-grams		
DE	autofahrer	fahr	fahrer	auto
	freundeskreis	kreis	kreis>	<freun
	grundwort	wort	wort>	grund
	sprachschule	schul	hschul	sprach
	tageslicht	licht	gesl	tages
EN	anarchy	chy	<anar	narchy
	monarchy	monarc	chy	<monar
	kindness	ness>	ness	kind
	politeness	polite	ness>	eness>
	unlucky	<un	cky>	nlucky
	lifetime	life	<life	time
	starfish	fish	fish>	star
	submarine	marine	sub	marin
	transform	trans	<trans	form
FR	finirais	ais>	nir	fini
	finissent	ent>	finiss	<finis
	finissions	ions>	finiss	sions>

Ejemplos de char n-grams frecuentes

FastText (notas importantes)

- FastText es rápido en entrenamiento (1B tokens ~ 10 mins in CPU).

- La librería proporciona vectores de palabras pero si hay OOV usa los n-grams para calcular el nuevo vector.

- FastText es invariante al orden de los tokens (BOW).

- FastText es independiente del contexto (vectores estáticos).