Recurrent Neural Networks (RNNs)

Michel RIVEILL
michel.riveill@univ-cotedazur.fr

Motivation

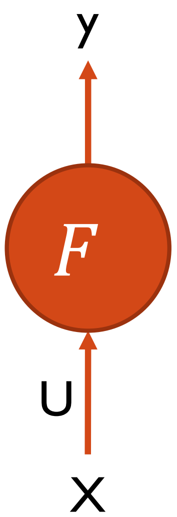
- Humans don’t start their thinking from scratch every second
  - Thoughts have persistence
- Traditional neural networks can’t characterize this phenomena
  - Ex: classify what is happening at every point in a movie
  - How a neural network can inform later events about the previous ones
- Recurrent neural networks address this issue
  - Some applications
    - NLP: Same word may have a different label depending on the context.
      - NER - Apple CEO Tim Cook eat an apple
      - POS - The bank is located on the bank (La banque est située sur la berge)
    - Forcasting - Time-series Prediction
- How?
  - Add state to artificial neurons

What are RNNs?

- Main idea is to make use of sequential information
- How RNN is different from neural network?
  - Vanilla neural networks (MLP) assume all inputs and outputs are independent of each other
  - But for many tasks, that's a very bad idea
- What RNN does?
  - Perform the same task for every element of a sequence
    - That's what recurrent stands for
  - Output depends on the previous computations!
- Another way of interpretation – RNNs have a "memory"
  - To store previous computations

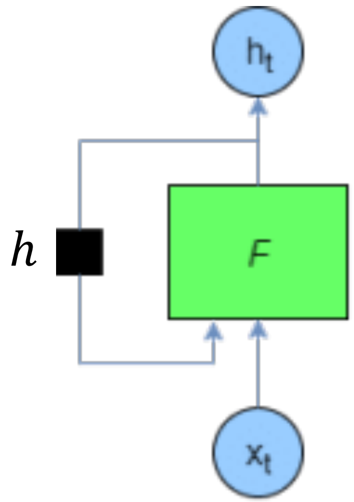
From vanilla NN to recurrent NN

- Vanilla cell
  - $y = F(U.X)$



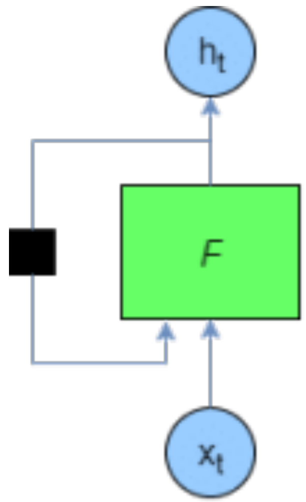
From vanilla NN to recurrent NN

- Vanilla cell
  - $y = F(U.X)$
- Recurrent cell → use 2 weights matrix
  - Add an internal variable: h
  - The output depends to the current entry and the previous internal variable:
    - $h_t = F(W.h_{t-1} + U.X_t)$
    - Could be rewritten on $h_t = F(V.[h_{t-1}, X_t])$



From vanilla NN to recurrent NN

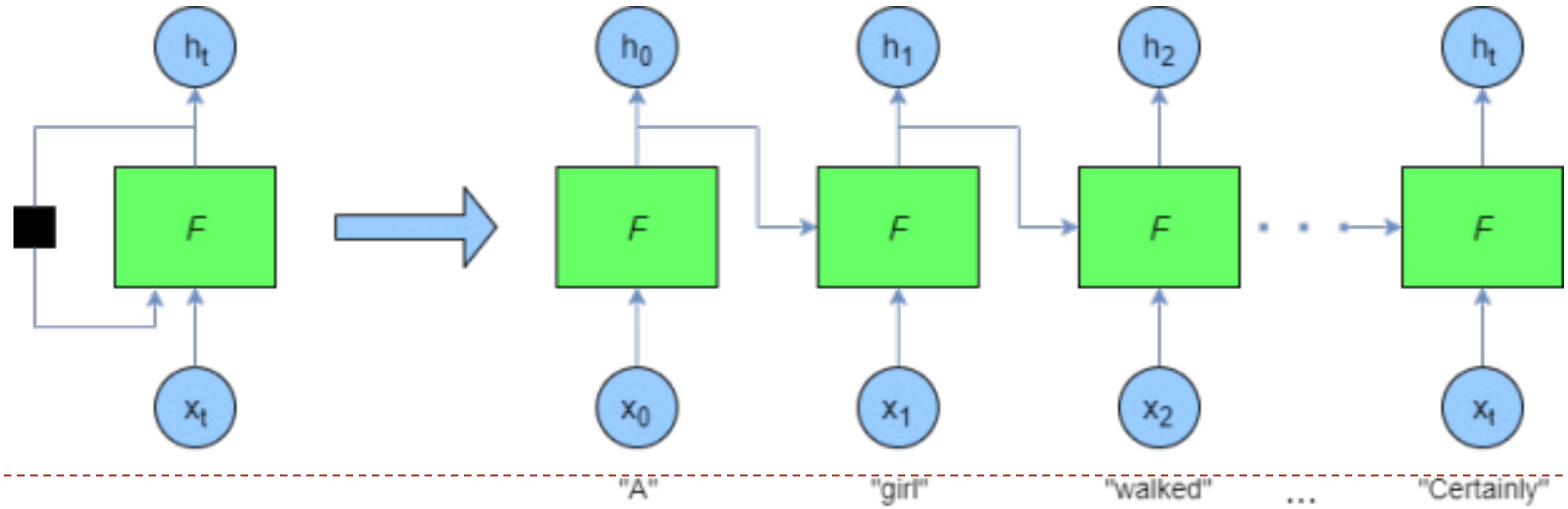
- Vanilla cell
  - $y = F(U.X)$
- Recurrent cell → use for each time step the **same weights matrix**
  - $h_t = F(W.[h_{t\,-\,1}, Xt])$
- Recurrent layer, step by step
  - at each time step
    - A new entry is being supplied
    - And a new output (h_t) is calculated using:
      - □ The new input X_t
      - □ The output of the previous step $h_{t\,-\,1}$



  - $h_1 = F(W.[h_0, X_1])$
  - $h_2 = F(W.h_1, X_2])$
  - $h_3 = F(W.h_2, X_3])$
  - …

From vanilla NN to recurrent NN

- Vanilla cell
  - $y = F(U.X)$
- Recurrent cell
  - $h_t = F(W.[h_{t\,-\,1}, X_t])$
- Recurrent neural networks are "unrolled" programmatically during training and prediction
  - All neurons share the **same weight matrix**

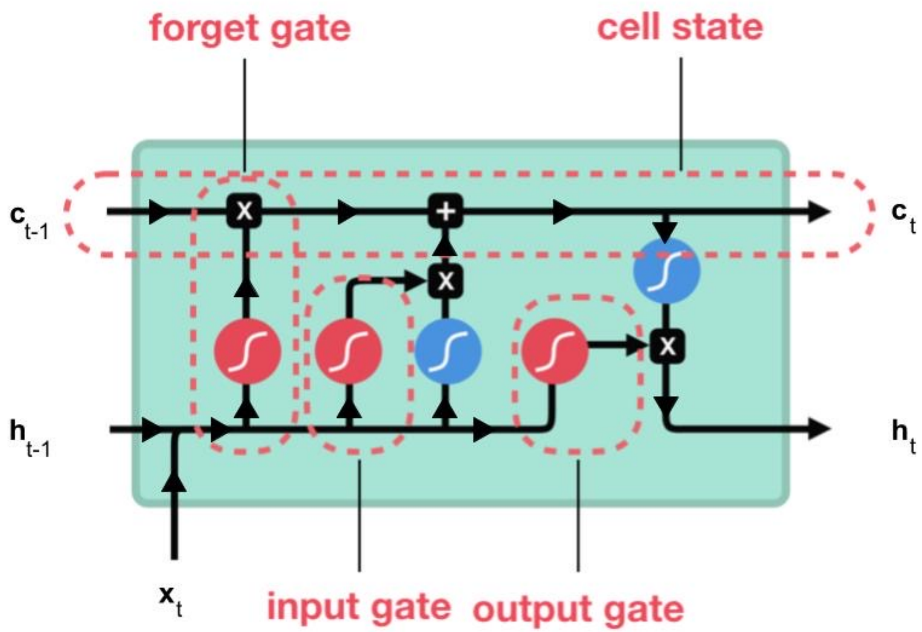


RNN from scratch

- If you want to fully understand how RNNs work, you can read
  - https://medium.com/nerd-for-tech/recurrent-neural-networks-3a0adb1d4515
- and analyse the code
  - https://github.com/maciejbalawejder/Deep-Learning-Collection/blob/main/Sequentials/RNN/RNN.ipynb

LSTM cell

- Contain three "gates":
  - Calculation zones that regulate the flow of information (by performing specific actions).
  - Forget gate (porte d'oubli)
  - Input gate (porte d'entrée)
  - Output gate (porte de sortie)
- h_t: Hidden state (état caché)
  - The eventual output
- c_t: Cell state (état de la cellule)
  - Like residual



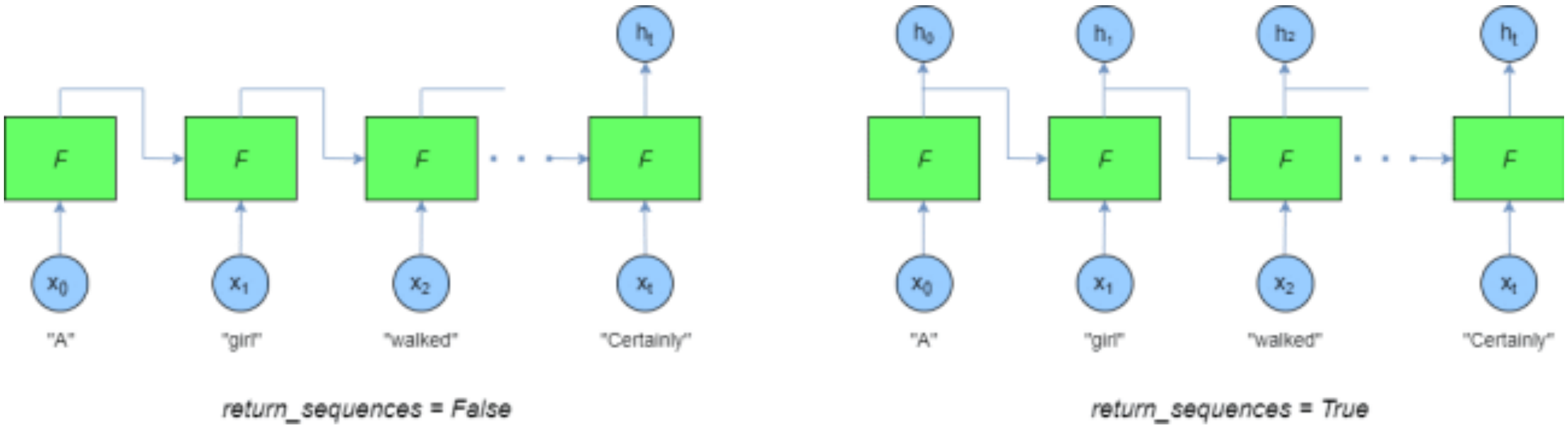
(crédit : image modifiée de Michaël Nguyen)

Keras Long Short-Term Memory Cell

- **from** tf.keras.layers **import** LSTM

- Main params
  - **Units:** dimension of output space

  - **return_sequences:** True or False
    - If False return only the last output
    - If True return the full sequence of the output sequence
      - Output sequence = hidden state (the vocabulary change regardind documentation)
  - **return_state**: True or False
    - If True return 3 values
      - The full output sequence or only the last one (depend on return_sequences)
      - The last output sequence
      - The cell state
    - If False return nothing
  - **stateful**: True or **False**.
    - If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.
      - You have to put shuffle=False in a fit method
    - **Nobody uses statefull LSTM**


F
h
x0
x1
x2
xt
h0
h1
h2
ht
"A"
"girl"
"walked"
"Certainly"
return_sequences = False
return_sequences = True

If there's a Dense after LSTM. How many dense cells are used?

- X = LSTM(500, return sequence = False or True)
- Output = Dense(…)(X)

With return_sequences=False,

Dense layer is applied only once at the last cell

With return_sequences=True

Dense layer is applied to every timestep

Keras Gated Recurrent Unit Cell

- **from** keras.layers **import** GRU

- Main params (similar to LSTM)
  - **Units:** dimension of output space

  - **return_sequences:** True or False
    - If False return only the last output
    - If True return the full sequence of the output
      - Output sequence = hidden state (the vocabulary change regardind documentation)
  - **return_state**: True or False
    - If True return 2 values
      - The full output sequence or only the last one (depend on return_sequences)
      - The last output sequence (is equivalent to the celle state for a GRU)
    - If False return nothing
  - **stateful**: True or False
    - If True, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.
      - You have to put shuffle=False in a fit method
    - **Nobody uses statefull GRU**

A basic example: sentiment analysis or text classification

- inputs = Input(shape=(1,)) **# Input size are unknow**
  - **Output shape=[None, 1]**
- tokens = Text_vectorizer(inputs)
  - **Output shape=[None, max_len]**
- embedding = Embedding(VOCABULARY_SIZE, EMBEDDING_SIZE)(tokens)
  - **Output shape=[None, max_len, EMBEDDING_SIZE]**
- outputs = LSTM(**16**, return_sequences=False)(embedding)
  - **Output shape=[None, 16]**
- predictions = Dense(nb_classes, activation='softmax')(outputs)

- model = Model(inputs, predictions)

- Fit by batch
  - model.fit(X, y, ….)

A basic example: forcasting → predict next time step

inputs = Input(shape=(None,1)) **# Input size are unknow**

- **Output shape=[None, None,1]**

output = LSTM(**16**, return_sequences=False)(embedding)

- **Output shape=[None, 16]**

predictions = Dense(1, activation='linear')(output)

- **Output shape=[None, 1]**

- model = Model(inputs, predictions)

- Fit by batch
  - Model.fit(X, y, ….)

RNN in NLP task

Some use of RNN
→ Text Classification / Sentiment analysis

Affect a label to a text

- Classify a
  - restaurant review from Yelp!
  - movie review from IMDB

  …

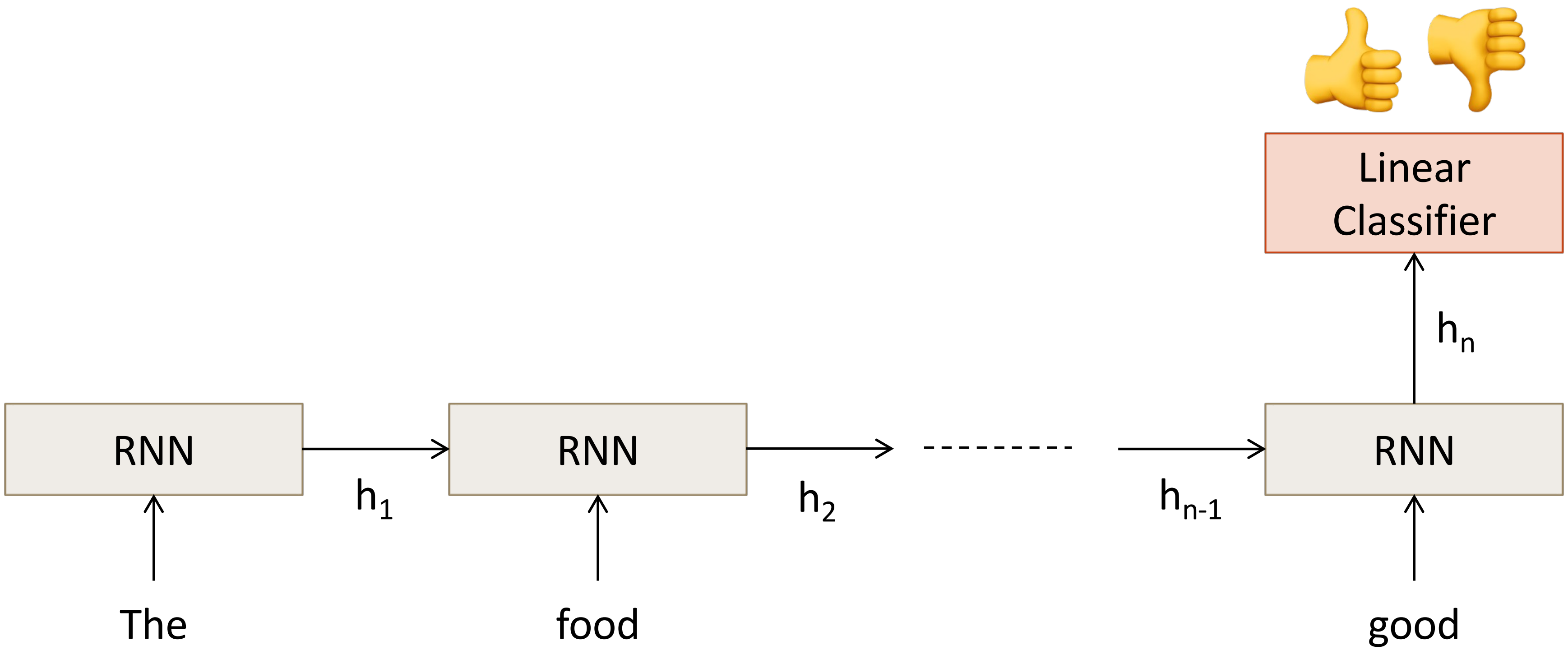
  as positive or negative

- Inputs:
  - Multiple words, one or more sentences
- Outputs:
  - Positive / Negative classification

- "The food was really good"
- "The chicken crossed the road because it was uncooked"

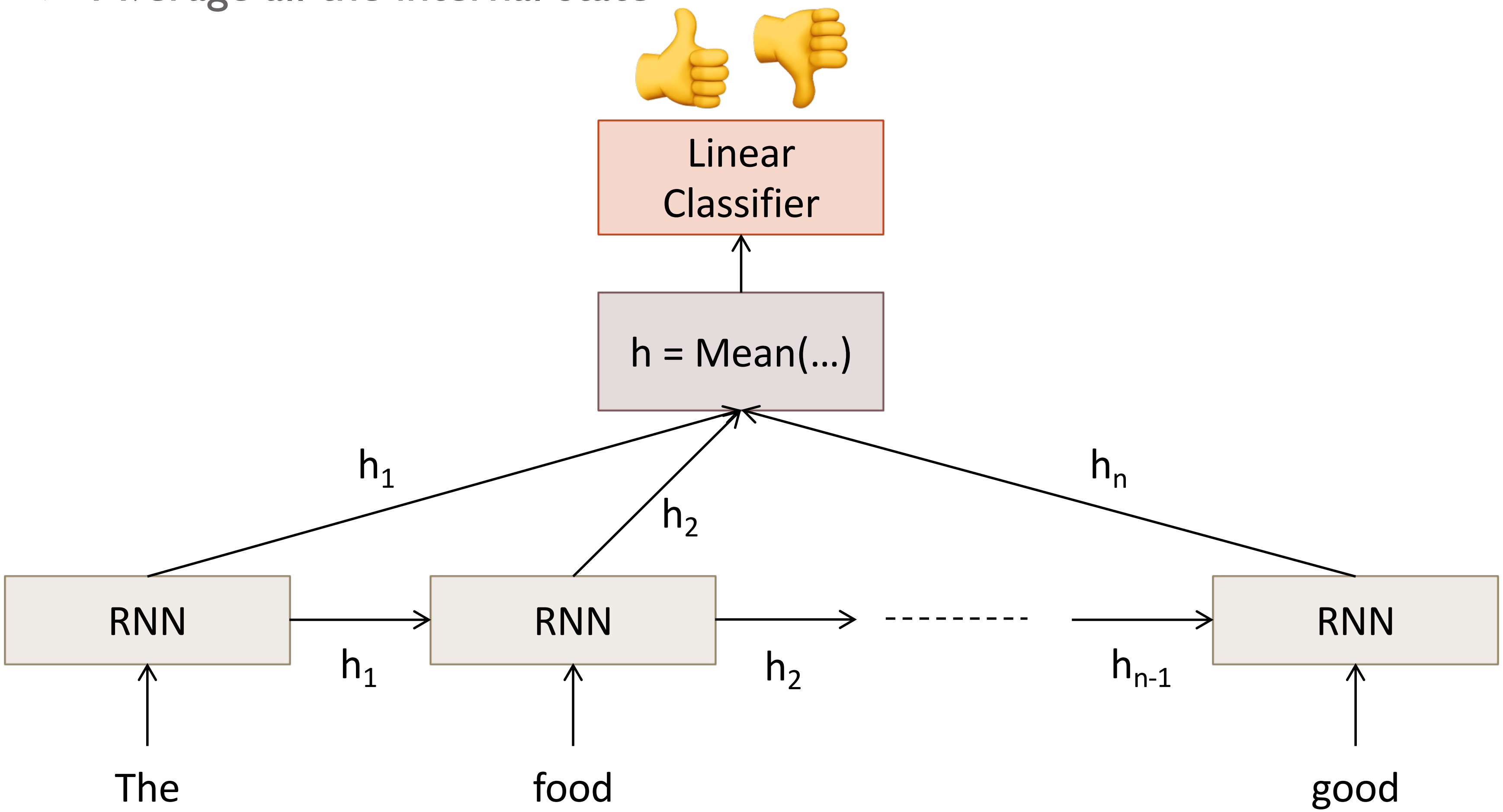
Sentiment analysis - solution 1

- retrieve only the last state



Sentiment analysis – solution2

- Other possible architecture
  - Average all the internal state

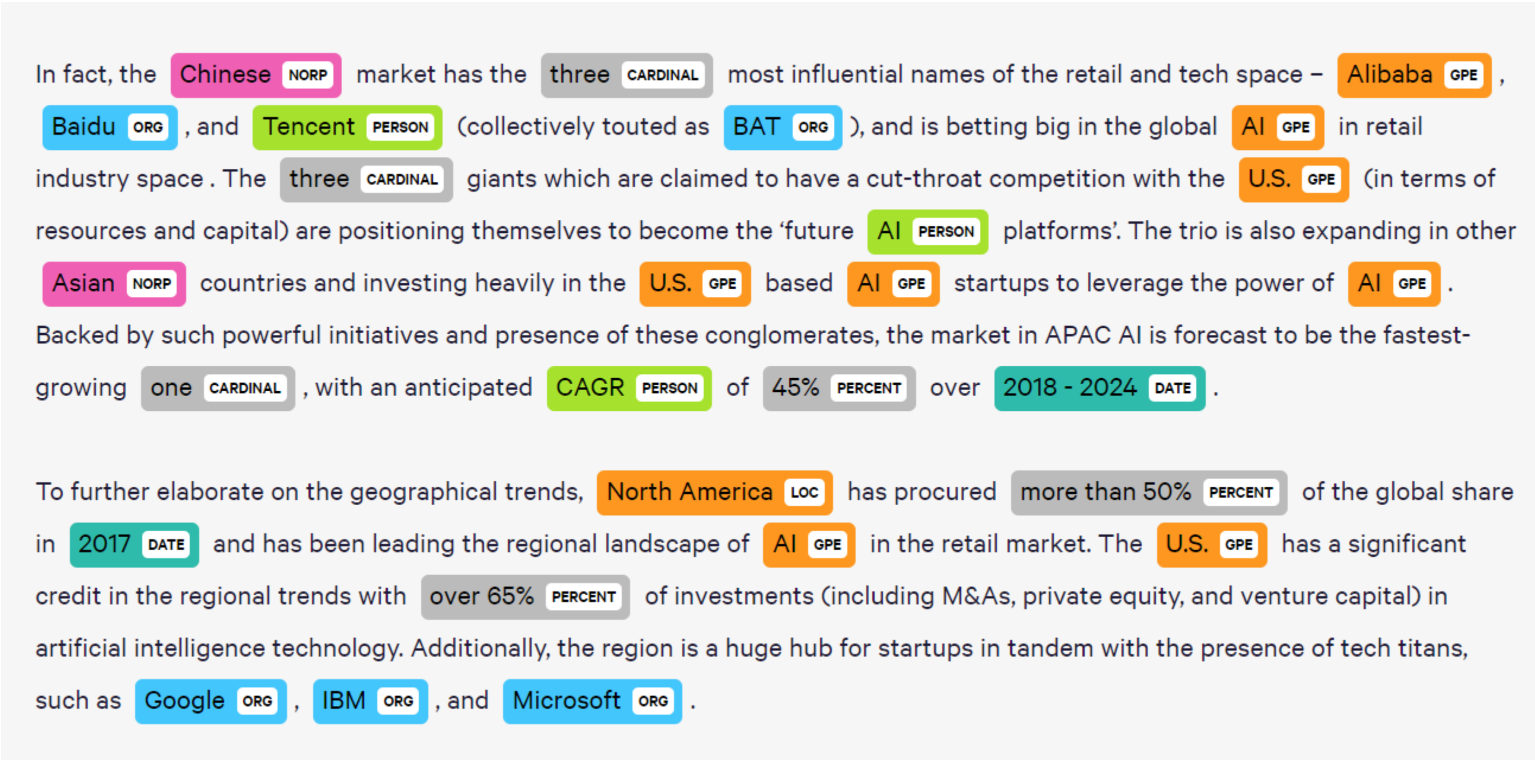


http://deeplearning.net/tutorial/lstm.html

Some use of RNN

→ Named Entity Recognition / Part of Speech Tagging

- Affect a label to each word
  - **find** and **classify** names in text
    - Could bean entity : number, country, person, … (NER)
    - Could be a function : noun, verb, adverbs, … (POS)



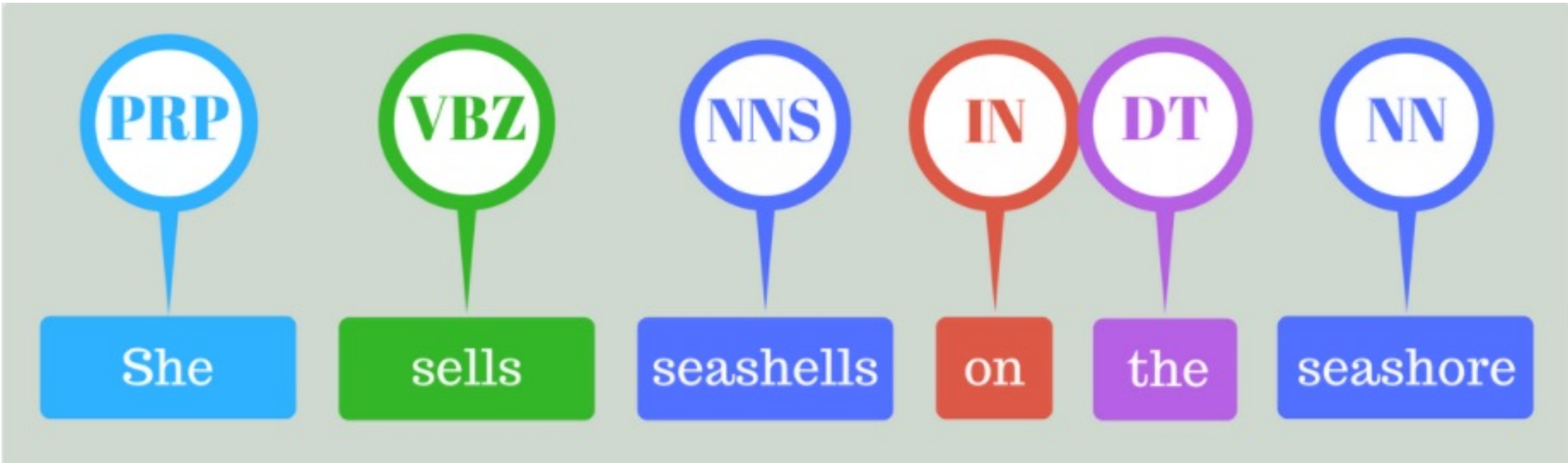
Extract information from text

Vente Villa 4 pièces Nice (06000)
Réf. 12390: Sur les Hauteurs de Nice. Superbe villa moderne (190m2), 2 chambres et 1 suite parentale, 3 salles de bain. Très grand salon/salle à manger, cuisine américaine équipée. Prestations de haut standing. Vue panoramique sur la mer. Cette villa a été construite en 2005. 1 270 000 euros. Si vous êtes intéressés, contactez vite Mimi LASOURIS 06.43.43.43. 43

REAL ESTATE TEMPLATE
Reference: 12390
Prize: 1 270 000
Surface: 190 m2
Year Built: 2005
Rooms: 4
Owner: Mimi LASOURIS
Telephone: 06.43.43.43. 43

Some use of RNN
→ Named Entity Recognition / Part of Speech Tagging

- Affect a label to each word
  - **find** and **classify** names in text
    - Could bean entity : number, country, person, … (NER)
    - Could be a function : noun, verb, adverbs, … (POS)



Named Entity Recognition

- NER = identification and categorization
  - Entities (persons, organizations, locations)
  - Times (dates, times and durations)
  - Quantities (monetary values, measures, percentages and cardinal numbers)
  - Depend of the domain

- NER evaluation: 2-by-2 contingency table

	Correct	Not correct
selected	TP	FP
Not selected	FN	TN

- Precision: % of selected items that are correct
- Recall: % of correct items that are selected
- F-measure: weighted harmonic mean

Label encoding for NER

	IO encoding	IOB encoding
Fred	PER	B-PER
showed	O	O
Sue	PER	B-PER
Mengqiu	PER	B-PER
Huang	PER	I-PER
's	O	O
new	O	O
painting	O	O

How to evaluate NER task: CoNLL03

- The Language-Independent Named Entity Recognition task introduced at CoNLL-2003 measures the performance of the systems in terms of precision, recall and f1-score, where:
  - *"precision is the percentage of named entities found by the learning system that are correct.*
  - *Recall is the percentage of named entities in the corpus found by the system.*
  - *A named entity is correct only if it is an exact match of the corresponding entity in the data file."*
- This approach is currently considered to be too unsophisticated, particularly when using BIO tags.

How to evaluate NER task: SemEval

- The SemEval'13 introduced four different ways to measure precision/recall/f1-score results based on the metrics defined by MUC.
  - **Strict**: exact boundary surface string match and entity type
  - **Exact**: exact boundary match over the surface string, regardless of the type
  - **Partial**: partial boundary match over the surface string, regardless of the type
  - **Type**: some overlap between the system tagged entity and the gold annotation is required

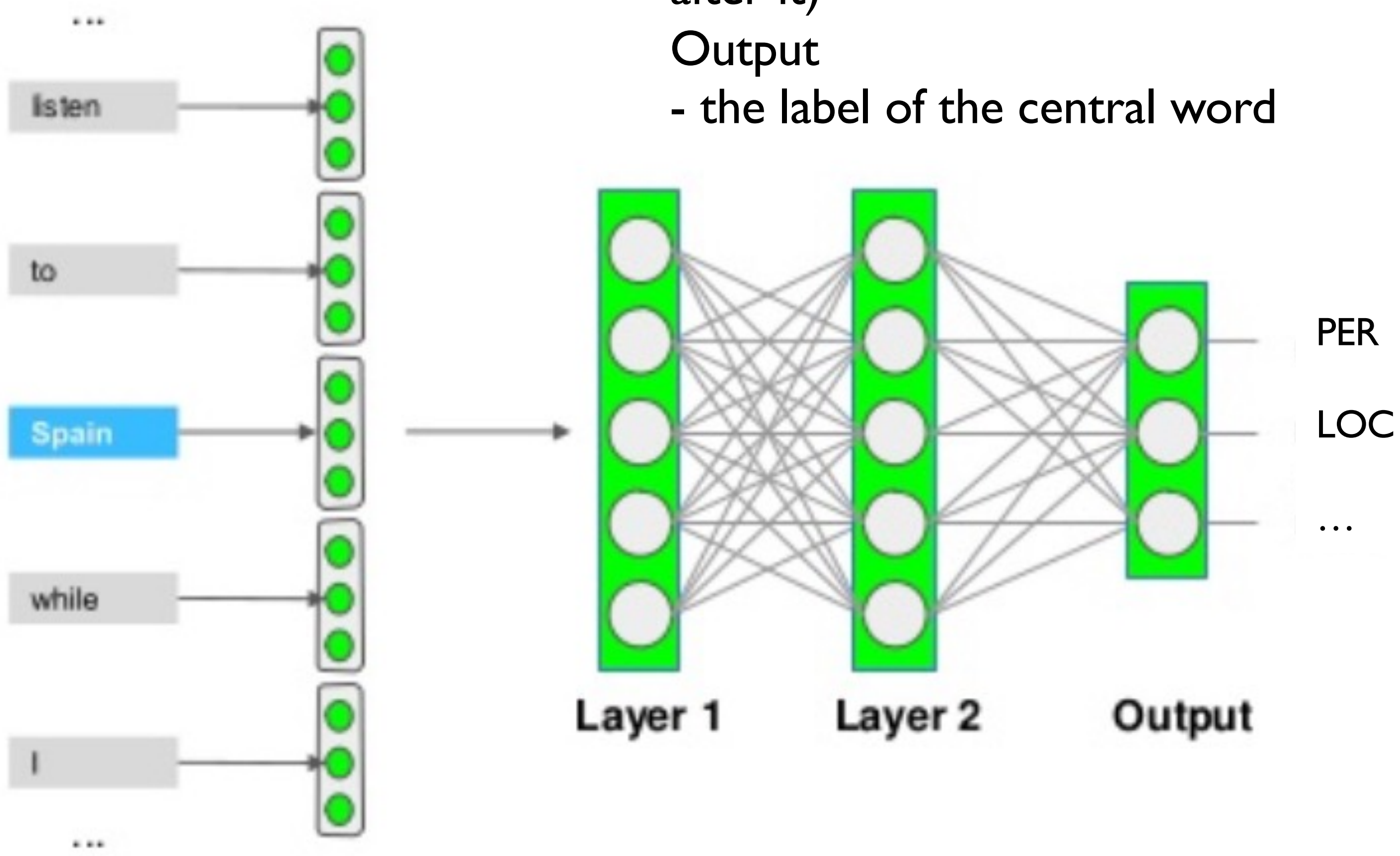
Implementation: https://github.com/chakki-works/seqeval

Machine Learning approach

- Features for sequence labeling
  - Words
    - Current word (essentially like a learned dictionary)
    - Previous/next word (context)
    - Other kinds of inferred linguistic classification – Part-of-speech tags
    -
  - Label context
    - Previous (and perhaps next) label

MLP for NER/POS

Input :
- a word and its context (the words before and after it)
Output
- the label of the central word

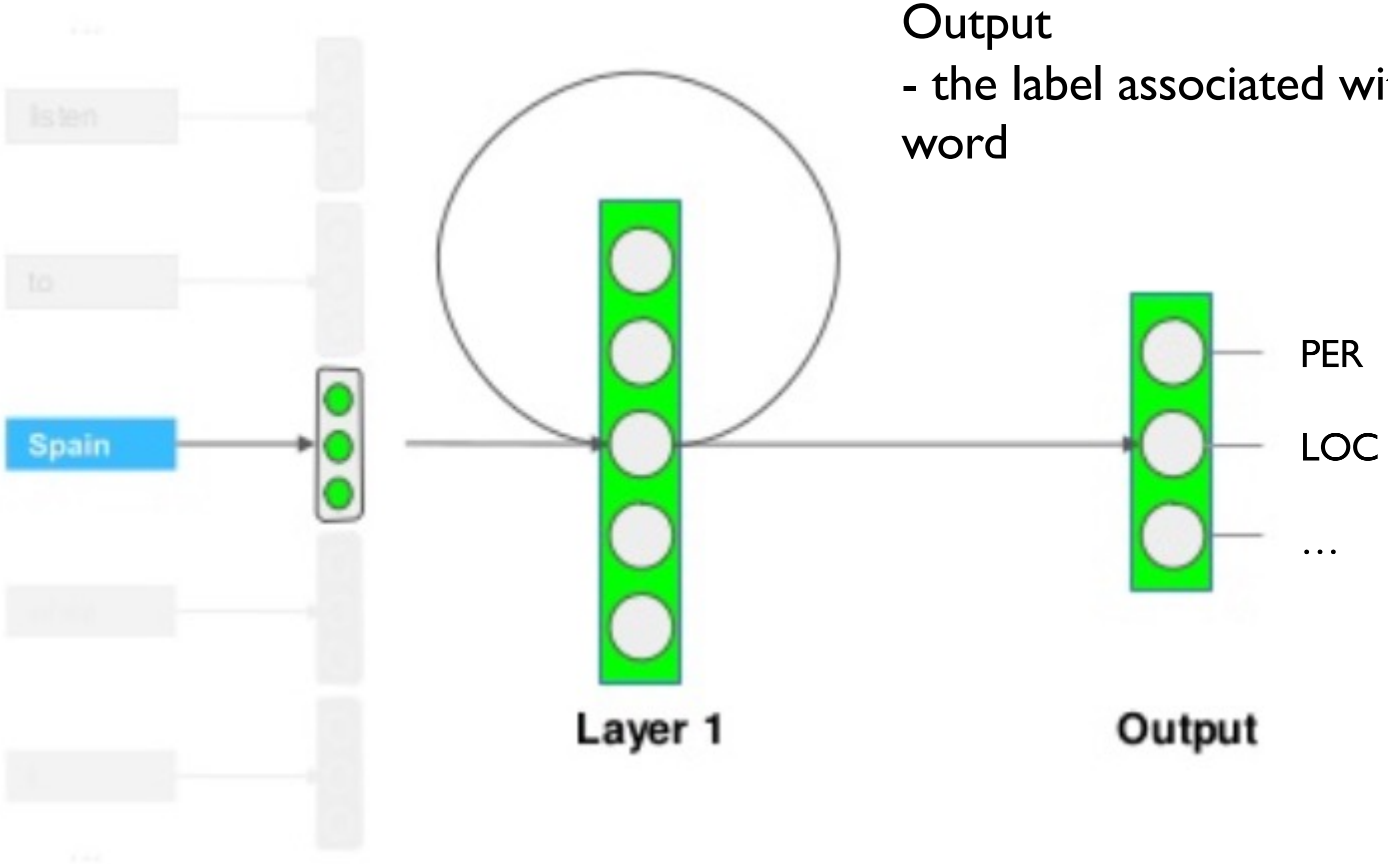


- Activation function for output: softmax

Recurrent neural network for NER

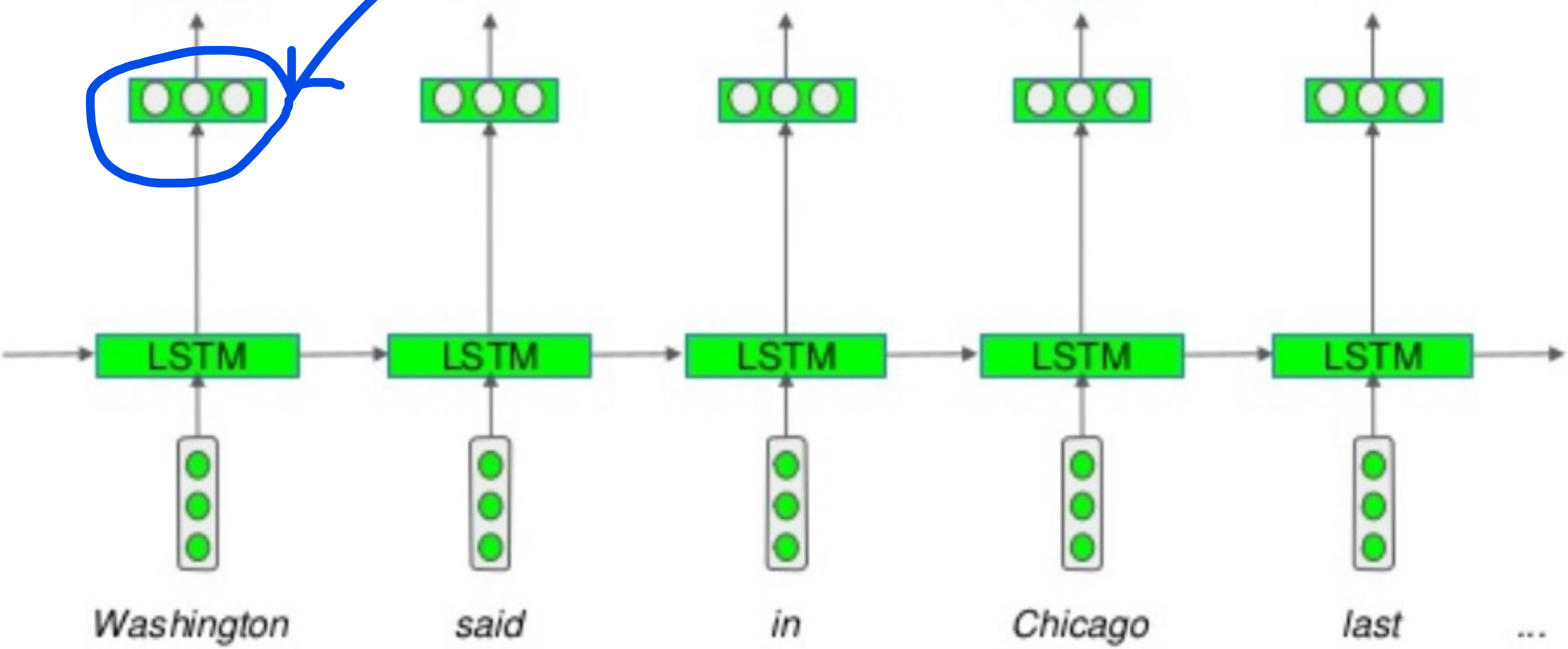
Input :
- a phrase (i.e. a list of words)

Output
- the label associated with each word



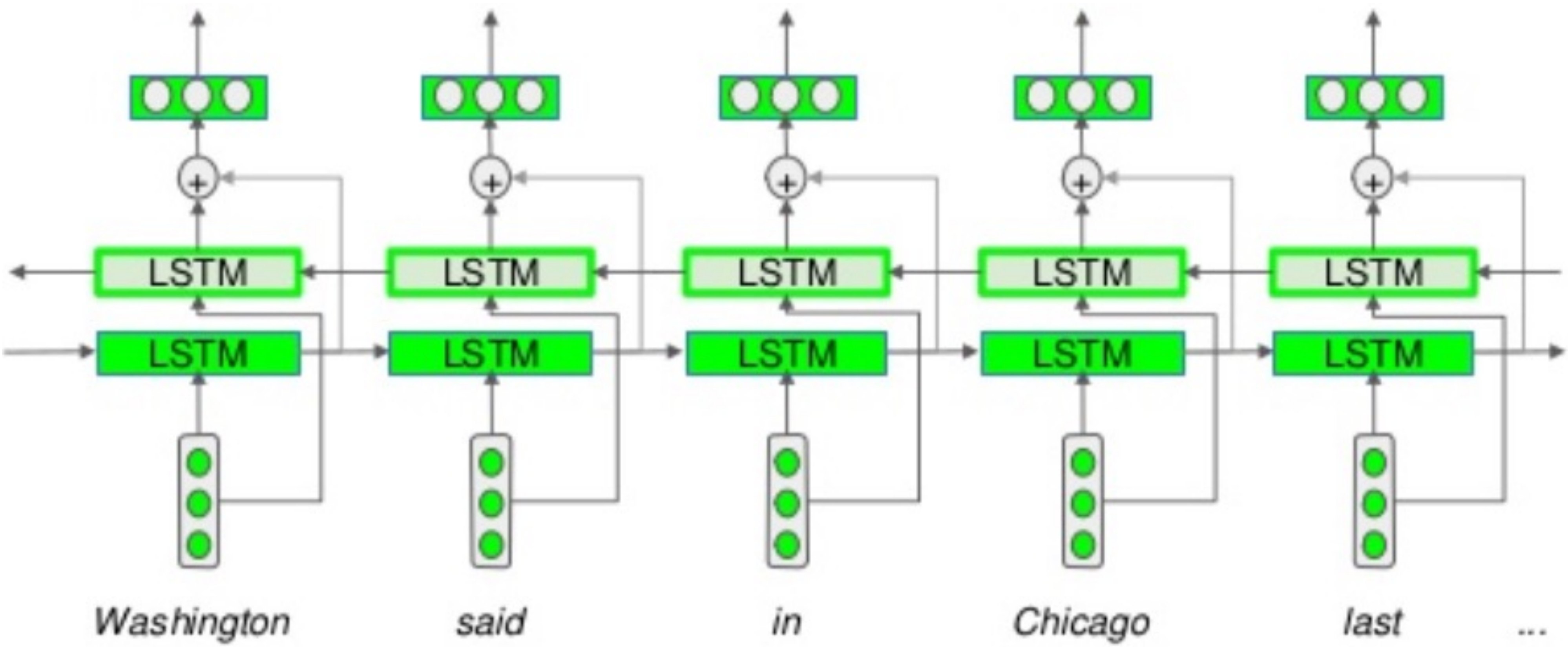
- Activation function for output: softmax

Recurrent neural network for NER (same network but unfolded)



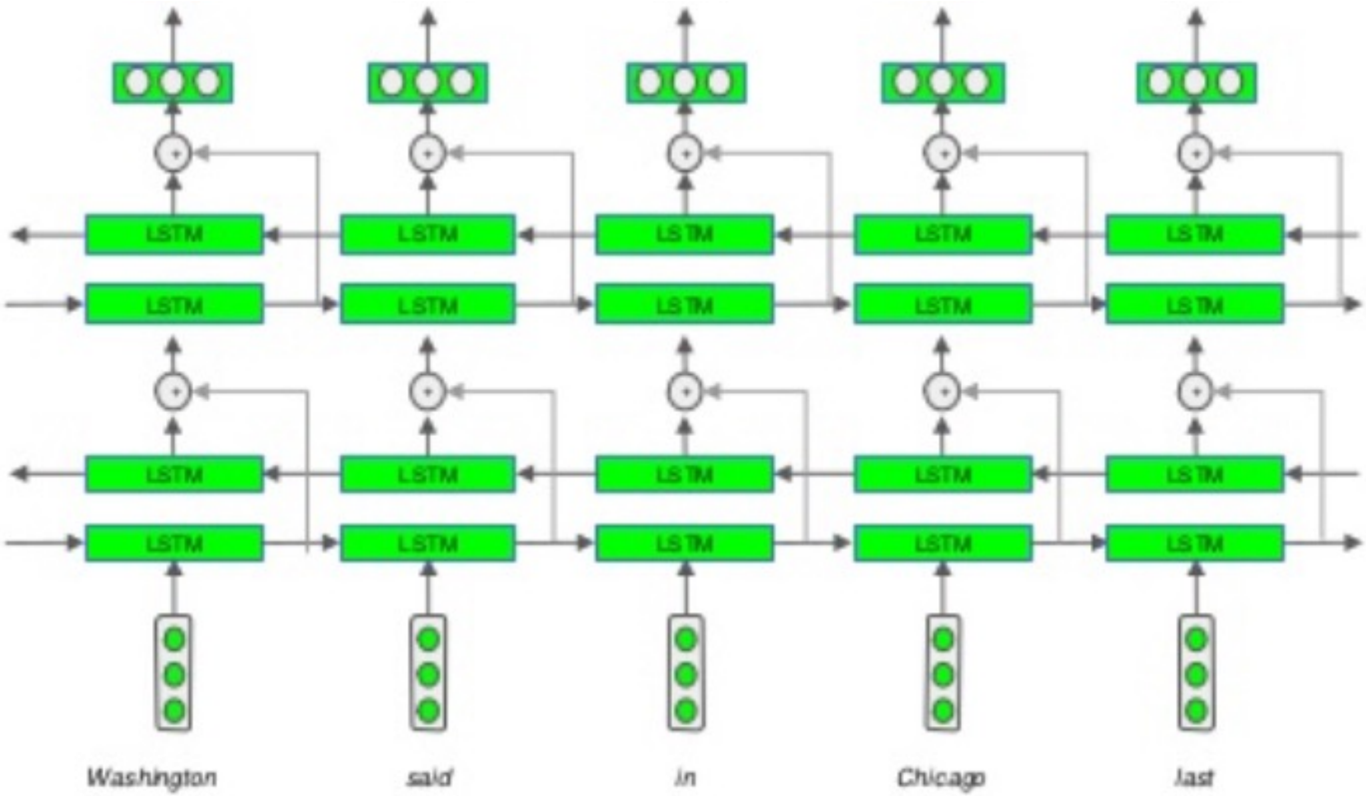
- Activation function for output: softmax

Bi directional recurrent neural network for NER



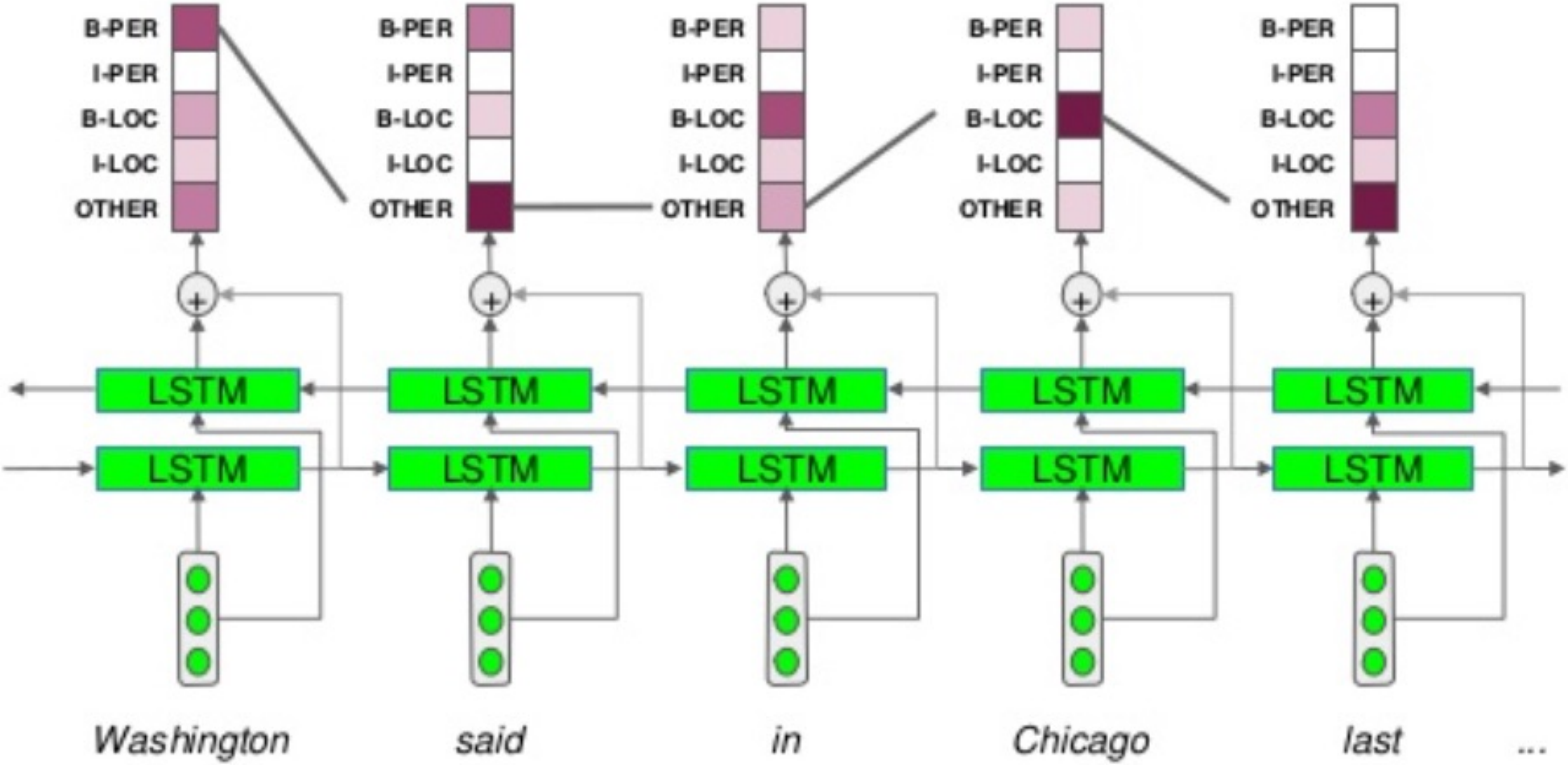
- Activation function for output: softmax

Stacked Bi-RNN



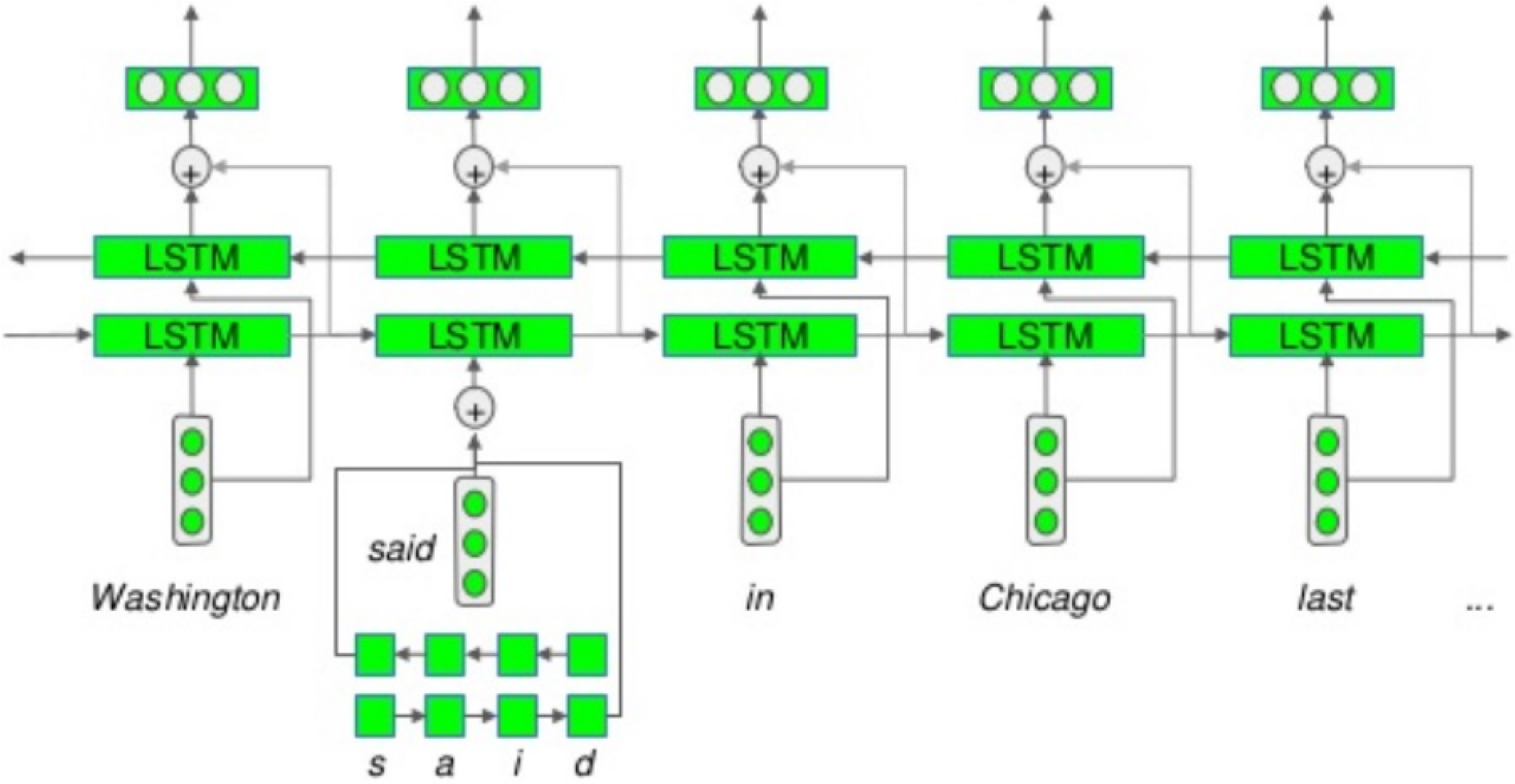
- Activation function for output: softmax

Bi-RNN + CRF



For example: https://www.tensorflow.org/addons/api_docs/python/tfa/layers/CRF

Multi-level encoding
char encoding + word encoding



TODAY Lab

1. Sentiment analysis task
   - Use the same data set as in the machine learning course
   - Build a recurrent network to predict sentiment
   - Did you obtain a better score than with a machine learning approach?
   - Evaluate your model with F1 score

Optional

2. Naming Entity Recognition task
   - Annotate with Spacy some text for a NER task
   - **Build a baseline model (Machine learning) to predict NER**
     - If X is a pandas with a list of word, you can obtain the one hot encoded representation of X by
       - v = sklearn.feature_extraction.DictVectorizer(sparse=False)
       - X = v.fit_transform(X.to_dict('records'))
     - y is a label associated to the words
     - Use for example Logistic Regression model
   - **Build a stacked bi-LSTM network to predict NER**
   - Evaluate your result with seqeval partial mode

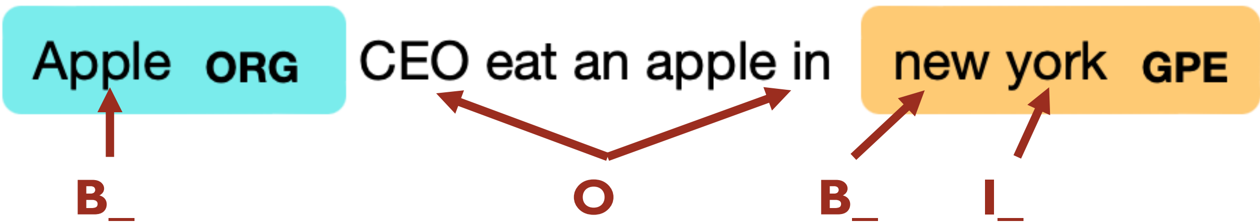
Same as usual: Upload a notebook on lms

How to annotate text with Spacy

```
!pip install -U spacy
import spacy
from spacy import displacy

!python -m spacy download en_core_web_sm
nlp = spacy.load("en_core_web_sm")

annotated_text = nlp("Apple CEO eat an apple in new york")
displacy.render(annotated_text, style="ent", jupyter=True)
```



Corpus annotation with Spacy

```
import pandas as pd
from nltk.corpus import reuters # you can use another corpus

text = []
for file in reuters.fileids():
    text += [reuters.raw(file)]

docs, words, labels = [], [], []
for i, doc in enumerate(text[:1000]): # prendre uniquement 1000 texte
    annotated_doc = nlp(doc)
    for ent in annotated_doc:
        words += [ent.text]
        if ent.ent_iob_!="O":
            labels += [ent.ent_iob_+"_"+ent.ent_type_]
        else:
            labels += [ent.ent_iob_]
    docs += [i]*len(annotated_doc)

df = pd.DataFrame({"doc":docs, "words":words, "labels":labels})
df.head()
```

Common remark about this lab

- Sentiment analysis task
  - The last output can be taken → return_sequences=False
  - Outputs can be averaged → return_sequences=True
    - mean = GlobalAveragePooling1D layer
    - But which axis ?
      - Output = None, lstm_size or None, max_len ?
      - Correct answer: None, lstm_size
    - How to test ?
      - Print the shape…
        - x = GlobalAveragePooling1D(axis=2)(x)
        - print(x.shape)

Common remark about this lab

- Naming Entity Recognition task
  - Use only one tokenizer, especially when you want to associate a label with each token.
    - Otherwise, the labels have to be split in the same way as the input vector.
  - The translation "token" → "index" can be made with a dictionary built on all the words of train or external sources
    - Beware of data leakage:
    - Parameter of TextVectorizer layer (no need to adapt if you pass a dict)

Layer TimeDistributed useless with tensorflow 2. after Dense

- There was none in my slides.... And it's not a mistake on my part
  - What does the documentation say?
  - What should I check?
  - What does ChatGPT think?

- Documentation
  - This wrapper allows to apply **the same layer** to every temporal slice of an input.
  - Every input should be at least 3D, and the dimension of index one of the first input will be considered to be the temporal dimension.
    - Consider a batch of 32 video samples, where each sample is a 128x128 RGB image with channels_last data format, across 10 timesteps. The batch input shape is (32, 10, 128, 128, 3).
    - You can then use TimeDistributed to apply the same Conv2D layer to each of the 10 timesteps, independently:

- Summary
  - Cells != Dense → use TimeDistributed (to reuse the same cells if return_sequence=True)
  - Cells == Dense
    - After 2020 (tensorflow 2.0) → TimeDistributed is not necessary for Dense
      - If you use or not, Dense is not replicated
    - Before 2020 (tensordlow 1.0) → TimeDistributed is mandatory
      - If you don't use: Dense is replicated

TimeDistributed or not: what should be checked?

One or more Dense cells after LSTM/GRU?

- **One cell:** obviously the same for each "time step"

An experiment to check

- inputs = layers.Input(input_shape=(MaxLen, InputSize))
- x = GRU(HiddenSize, return_sequences=True)(inputs)
- outputs1 = TimeDistributed(Dense(OutputSize), activation= 'softmax')(x)
- outputs2 = Dense(OutputSize), activation= 'softmax')(x)
- x3 = GRU(HiddenSize, return_sequences=False)(inputs)
- outputs3 = Dense(OutputSize), activation= 'softmax')(x3)
- model1 = Model(inputs, outputs1)
- model2 = Model(inputs, outputs2)
- model3 = Model(inputs, outputs3)

```
Layer (type)                 Output Shape              Param #
=================================================================
gru_3 (GRU)                  (None, 64, 16)            1584

time_distributed_1 (TimeDi   (None, 64, 8)             136
stributed)

=================================================================
Total params: 1720 (6.72 KB)
```

One or more dense? 8*16+8 = 136

```
Layer (type)                 Output Shape              Param #
=================================================================
gru_4 (GRU)                  (None, 64, 16)            1584

dense_4 (Dense)              (None, 64, 8)             136

=================================================================
Total params: 1720 (6.72 KB)
```

```
Layer (type)                 Output Shape              Param #
=================================================================
gru_6 (GRU)                  (None, 16)                1584

dense_6 (Dense)              (None, 8)                 136

=================================================================
Total params: 1720 (6.72 KB)
```

TimeDistrinuted: what ChatGPT says?

Faut-il un timedistributed après LSTM avec des cellules Dense en Tensorflow ?

- Si vous avez une tâche de prédiction de séquence, vous pouvez utiliser une couche Dense après la couche LSTM pour effectuer la prédiction. Cependant, la sortie de la couche LSTM est tridimensionnelle (batch_size, timesteps, features) et la couche Dense attend une entrée bidimensionnelle (batch_size, features).
- Pour résoudre ce problème, vous pouvez utiliser la couche TimeDistributed. Elle permet d'appliquer la couche Dense à chaque étape de temps de la séquence de manière indépendante. Ainsi, vous obtenez une prédiction pour chaque étape de temps.

Es-tu sur de la réponse ? La documentation Tensorflow dit le contraire

- **Je m'excuse** pour la réponse précédente. Vous avez raison, et je vous remercie de votre correction. En général, lorsque vous utilisez une couche LSTM suivie de cellules Dense dans TensorFlow, il n'est généralement pas nécessaire d'utiliser TimeDistributed. La couche LSTM gère déjà la séquence temporelle, et la couche Dense qui suit peut accepter les sorties de la couche LSTM telles quelles.

Dictionnaire python

- d = {'apple':4, 'orange':5, 'pear':6}
- x = d['pineapple']

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[14], line 1
----> 1 x = d['pineapple']
      2 print(x)

KeyError: 'pineapple'
```

- x = d.get('pineapple')
- print("x =", x)
  - x = None
- x = d.get('pineapple', 12)
- print("x =", x)
  - x = 12