FNLP
Sequence Tagging II – Linear Models and Beyond

Yansong Feng
fengyansong@pku.edu.cn

Wangxuan Institute of Computer Technology
Peking University

March 26, 2025

Outline

Outline

Our No.1 Tagger – HMM POS Tagger

- We represent
  - a sentence of any length n: $x_1, x_2, x_3, ...x_n$
  - its corresponding POS tag sequence; $y_1, y_2, y_3, ...y_n$
- We care the joint probability of a sentence and its POS tag sequence:

$$p(x_1, x_2, x_3, ...x_n, y_1, y_2, y_3, ...y_n)$$

(Generative Model)

- Then the most likely POS tag sequence for $x_1, x_2, x_3, ...x_n$:

$$\arg\max_{y_1...y_n} p(y_1, y_2, y_3, ...y_n) p(x_1, x_2, x_3, ...x_n | y_1, y_2, y_3, ...y_n)$$

- Make Markov Assumptions (e.g., Trigram)

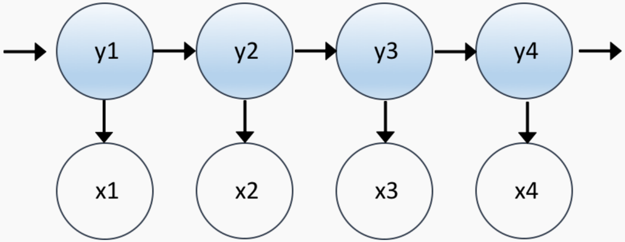
$$\arg\max_{y_1...y_n} \prod_i p(y_i | y_{i-2}, y_{i-1}) \prod_i p(x_i | y_i)$$

Elements in HMM POS Tagger

- Elements
  - a sequence of words
  - a sequence of POS tags
  - the beginning and end of a sentence
- Parameters
  - Sequences of POS tags
  - Co-occurrences of words and POS tags

Elements in HMM POS Tagger

- Elements
  - a sequence of words
  - a sequence of POS tags
  - the beginning and end of a sentence
- Parameters
  - Sequences of POS tags $\rightarrow$ transition probabilities ($p(y_n|y_{n-2}, y_{n-1})$)
  - Co-occurrences of words and POS tags $\rightarrow$ emission probabilities ($p(x_n|y_n)$)



Elements in HMM POS Tagger

- Elements
  - a sequence of words
  - a sequence of POS tags
  - the beginning and end of a sentence
- Parameters
  - Sequences of POS tags $\rightarrow$ transition probabilities ($p(y_n|y_{n-2}, y_{n-1})$)
  - Co-occurrences of words and POS tags $\rightarrow$ emission probabilities ($p(x_n|y_n)$)

Anything else useful?

- if the current word ending with *ing, ed, se, ly, ical*, or ....
- if the previous word is *the*
- if the next word is *.*
- ...

A Naive Way to Incorporate

...... many p_{ML}s

- $p_{ML}(POS_{w_i} = \text{VB}|w_i$ ending with *ing*$)$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i$ ending with *ed*$)$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i$ ending with *se*$)$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i$ ending with *ly*$)$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i$ ending with *ical*$)$
- $p_{ML}(POS_{w_i} = \text{VB}|w_{i-1} =$ *the*$)$
- $p_{ML}(POS_{w_i} = \text{VB}|w_{i+1} =$. (a period)$)$
- ...

A Naive Way to Incorporate

...... many p_{ML}s

- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{ing})$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{ed})$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{se})$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{ly})$
- $p_{ML}(POS_{w_i} = \text{VB}|w_i \text{ ending with } \textit{ical})$
- $p_{ML}(POS_{w_i} = \text{VB}|w_{i-1} = \textit{the})$
- $p_{ML}(POS_{w_i} = \text{VB}|w_{i+1} = \text{. (a period)})$
- ...

This gives you lots of λs to tune.

Alternative: A Classifier

How about using a classifier ?

I love Beijing .

Alternative: A Classifier

How about using a classifier ?

I love Beijing .

- make predictions for each word
- for each word:
    - extract various features regarding that word
    - find the best POS label for the word

Alternative: A Classifier

How about using a classifier ?

I love Beijing .

- make predictions for each word
- for each word:
  - extract various features regarding that word
  - find the best POS label for the word

Alternative: A Classifier

How about using a classifier ?

I love Beijing .

- make predictions for each word
- for each word:
  - extract various features regarding that word
  - find the best POS label for the word

Alternative: A Classifier

How about using a classifier ?

I love <u>Beijing</u> .

- make predictions for each word
- for each word:
    - extract various features regarding that word
    - find the best POS label for the word
- Anything more?

Alternative: A Classifier

How about using a classifier ?

I love Beijing .

- make predictions for each word
- for each word:
  - extract various features regarding that word
  - find the best POS label for the word
- Anything different from HMM?

Alternative: A Classifier

How about using a classifier ?

I love Beijing .

- make predictions for each word
- for each word:
  - extract various features regarding that word
  - find the best POS label for the word
- Anything different from HMM?
  - more features?

Alternative: A Classifier

How about using a classifier ?

I love Beijing .

- make predictions for each word
- for each word:
  - extract various features regarding that word
  - find the best POS label for the word
- Anything different from HMM?
  - more features?
  - individual decisions v.s. a sequence of decisions

Another View: Features

Features: pieces of evidences describing some aspects of observed data x, usually with respect to the predicted label y

- computer vision
  - the shape, color, texture, size.....of an object
  - other objects nearby, relative postions
  - number of objects available
  - ...
- natural language process, e.g., POS tagging
  - the target word itself, prefix, suffix, capital or not, ...
  - context: words before/after the target, their morphology
  - number of those indications
  - ...

Another View: Features

Features: pieces of evidences describing some aspects of observed data x, usually with respect to the predicted label y

- computer vision
  - the shape, color, texture, size.....of an object
  - other objects nearby, relative postions
  - number of objects available
  - dense features
  - ...
- natural language process, e.g., POS tagging
  - the target word itself, prefix, suffix, capital or not, ...
  - context: words before/after the target, their morphology
  - number of those indications
  - dense features
  - ...

Recall: Features in NLP

Features in NLP: pieces of evidences describing some aspects of observed data x with respect to the predicted label y, usually with the purpose of providing a conditional probability $p(y|x) \rightarrow$ discriminative models

Often

- A feature is a function $f_i(x, y) \in \mathcal{R}$
- more often, it is a binary or indicator function
- for example,

$$f_i(x,y) = \begin{cases} 1 & \text{if } x = \text{Beijing and } y\text{=NNP} \\ 0 & \text{otherwise} \end{cases}$$

- if we have m aspects to describe an instance, i.e., m features:
  - **a feature vector** for each instance, (x, y)
  - $[f_1(x, y_1), f_2(x, y_1), f_3(x, y_1), ..., f_m(x, y_1), f_1(x, y_2), ..., f_m(x, y_2), ...]$
  - $[\underline{1, 0, 0, ..., 1, 0,}0, 0, ..., 0, ...]$ when we evaluate y_1

Recall: Features in NLP

Features in NLP: pieces of evidences describing some aspects of observed data x with respect to the predicted label y, usually with the purpose of providing a conditional probability $p(y|x) \rightarrow$ discriminative models

Also

We may also want to introduce features that are slightly complex

- considering previous sub-decisions

$$f_j(x, y) = \begin{cases} 1 & \text{if previous tag is ADJ and } y\text{=NNP} \\ 0 & \text{otherwise} \end{cases}$$

Outline

Feature-based Discriminative Models

A linear classifier with the form like, $\lambda_{f(x,y)} f(x,y)$, where λs are weights,

- build a linear function to map input x to label y
- possibly need a weight $\lambda_{f_i(x,y)}$ for each feature $f_i(x,y)$
- then, for each possible label y of instance x, we can compute a score:

$$score(x,y) = \sum_i \lambda_{f_i(x,y)} f_i(x,y)$$

- the classifier should choose y^*:

$$y^* = \arg\max_y \sum_i \lambda_{f_i(x,y)} f_i(x,y)$$

Feature-based Discriminative Models

A linear classifier with the form like, $\lambda_{f(x,y)} f(x,y)$, where λs are weights,

- build a linear function to map input x to label y
- possibly need a weight $\lambda_{f_i(x,y)}$ for each feature $f_i(x,y)$
- then, for each possible label y of instance x, we can compute a score:

$$score(x,y) = \sum_i \lambda_{f_i(x,y)} f_i(x,y)$$

- the classifier should choose y^*:

$$y^* = \arg\max_y \sum_i \lambda_{f_i(x,y)} f_i(x,y)$$

That is, for each y, compute the score, and select the y^* that gives the largest score.

Feature-based Discriminative Models

A linear classifier with the form like, $\lambda_{f(x,y)} f(x,y)$, where λs are weights,

- build a linear function to map input x to label y
- possibly need a weight $\lambda_{f_i(x,y)}$ for each feature $f_i(x,y)$
- then, for each possible label y of instance x, we can compute a score:

$$score(x,y) = \sum_i \lambda_{f_i(x,y)} f_i(x,y)$$

- the classifier should choose y^*:

$$y^* = \arg\max_y \sum_i \lambda_{f_i(x,y)} f_i(x,y)$$

The KEY: figure out those λs?

Feature-based Discriminative Models

A linear classifier with the form like, $\lambda_{f(x,y)} f(x,y)$, where λs are weights,

- build a linear function to map input x to label y
- possibly need a weight $\lambda_{f_i(x,y)}$ for each feature $f_i(x,y)$
- then, for each possible label y of instance x, we can compute a score:

$$score(x,y) = \sum_i \lambda_{f_i(x,y)} f_i(x,y)$$

- the classifier should choose y^*:

$$y^* = \arg\max_y \sum_i \lambda_{f_i(x,y)} f_i(x,y)$$

The KEY: figure out those λs? $\rightarrow$ We did this before in **Log-linear Models.**

Feature-based Linear Models: An Example

Tagging *Beijing* with a trained model:

I love Beijing .

- Clues: the target word, previous words, suffix, prefix, capitalized, ...
- **curwd_Beijing_, pre1wd_love_, pref_Be_, cap_1_, ...**

Feature-based Linear Models: An Example

Tagging *Beijing* with a trained model:

I love Beijing .

- Clues: the target word, previous words, suffix, prefix, capitalized, ...
- **curwd_Beijing_, pre1wd_love_, pref_Be_, cap_1_, ...**
- for each possible label (NNP, VB, ...), coupling clues with labels:

Feature-based Linear Models: An Example

Tagging *Beijing* with a trained model:

I love Beijing .

- Clues: the target word, previous words, suffix, prefix, capitalized, ...
- **curwd_Beijing_, pre1wd_love_, pref_Be_, cap_1_, ...**
- for each possible label (NNP, VB, ...), coupling clues with labels:
  - curwd_Beijing_**NNP**, pre1wd_love_**NNP**, pref_Be_**NNP**, cap_1_**NNP**...
  - curwd_Beijing_**VB**, pre1wd_love_**VB**, pref_Be_**VB**, cap_1_**VB**...

Feature-based Linear Models: An Example

Tagging *Beijing* with a trained model:

I love Beijing .

- Clues: the target word, previous words, suffix, prefix, capitalized, ...
- **curwd_Beijing_, pre1wd_love_, pref_Be_, cap_1_, ...**
- for each possible label (NNP, VB, ...), coupling clues with labels:
  - curwd_Beijing_**NNP**, pre1wd_love_**NNP**, pref_Be_**NNP**, cap_1_**NNP**...
  - curwd_Beijing_**VB**, pre1wd_love_**VB**, pref_Be_**VB**, cap_1_**VB**...
- obtain λs using certain algorithms,
  $\lambda_{\text{curwd\_Beijing\_NNP}} = 10$, $\lambda_{\text{pref\_Be\_NNP}} = 5$, $\lambda_{\text{cap\_1\_DT}} = -10$, ...

Feature-based Linear Models: An Example

Tagging *Beijing* with a trained model:

I love Beijing .

- Clues: the target word, previous words, suffix, prefix, capitalized, ...
- **curwd_Beijing_, pre1wd_love_, pref_Be_, cap_1_, ...**
- for each possible label (NNP, VB, ...), coupling clues with labels:
  - curwd_Beijing_**NNP**, pre1wd_love_**NNP**, pref_Be_**NNP**, cap_1_**NNP**...
  - curwd_Beijing_**VB**, pre1wd_love_**VB**, pref_Be_**VB**, cap_1_**VB**...
- obtain λs using certain algorithms,
  $\lambda_{\text{curwd\_Beijing\_NNP}} = 10$, $\lambda_{\text{pref\_Be\_NNP}} = 5$, $\lambda_{\text{cap\_1\_DT}} = -10$, ...
- compute: score(Beijing, NNP), score(Beijing, VB), score(Beijing, DT), ...

Feature-based Linear Models: An Example

Tagging *Beijing* with a trained model:

I love Beijing .

- Clues: the target word, previous words, suffix, prefix, capitalized, ...
- **curwd_Beijing_, pre1wd_love_, pref_Be_, cap_1_, ...**
- for each possible label (NNP, VB, ...), coupling clues with labels:
  - curwd_Beijing_**NNP**, pre1wd_love_**NNP**, pref_Be_**NNP**, cap_1_**NNP**...
  - curwd_Beijing_**VB**, pre1wd_love_**VB**, pref_Be_**VB**, cap_1_**VB**...
- obtain λs using certain algorithms,
  $\lambda_{\text{curwd\_Beijing\_NNP}} = 10$, $\lambda_{\text{pref\_Be\_NNP}} = 5$, $\lambda_{\text{cap\_1\_DT}} = -10$, ...
- compute: score(Beijing, NNP), score(Beijing, VB), score(Beijing, DT), ...
- choose the largest one: **score(Beijing, NNP)**

Feature-based Linear Models: An Example

Tagging *Beijing* with a trained model:

I love Beijing .

- Clues: the target word, previous words, suffix, prefix, capitalized, ...
- **curwd_Beijing_, pre1wd_love_, pref_Be_, cap_1_, ...**
- for each possible label (NNP, VB, ...), coupling clues with labels:
  - curwd_Beijing_**NNP**, pre1wd_love_**NNP**, pref_Be_**NNP**, cap_1_**NNP**...
  - curwd_Beijing_**VB**, pre1wd_love_**VB**, pref_Be_**VB**, cap_1_**VB**...
- obtain λs using certain algorithms,
  $\lambda_{\text{curwd\_Beijing\_NNP}} = 10$, $\lambda_{\text{pref\_Be\_NNP}} = 5$, $\lambda_{\text{cap\_1\_DT}} = -10$, ...
- compute: score(Beijing, NNP), score(Beijing, VB), score(Beijing, DT), ...
- choose the largest one: **score(Beijing, NNP)**
- tag *Beijing* with **NNP**

Feature-based Linear Models: An Example

Tagging *Beijing* with a trained model:

I love Beijing .

- Clues: the target word, previous words, suffix, prefix, capitalized, ...
- **curwd_Beijing_, pre1wd_love_, pref_Be_, cap_1_, ...**
- for each possible label (NNP, VB, ...), coupling clues with labels:
  - curwd_Beijing_**NNP**, pre1wd_love_**NNP**, pref_Be_**NNP**, cap_1_**NNP**...
  - curwd_Beijing_**VB**, pre1wd_love_**VB**, pref_Be_**VB**, cap_1_**VB**...
- obtain λs using **certain algorithms**,
  $\lambda_{\text{curwd\_Beijing\_NNP}} = 10$, $\lambda_{\text{pref\_Be\_NNP}} = 5$, $\lambda_{\text{cap\_1\_DT}} = -10$, ...
- compute: score(Beijing, NNP), score(Beijing, VB), score(Beijing, DT), ...
- choose the largest one: **score(Beijing, NNP)**
- tag *Beijing* with **NNP**

Features based Linear Models: Algorithms

The key is to choose proper weights λs for features

- the Perceptron algorithm
- Margin-based models (the Support Vector Machines, SVM)
- Exponential Models:
    - log-linear models, maximum entropy models, logistic models, ...
    - basically, produce a probabilistic model according to score(x, y)

$$p(y|x) = \frac{\exp score(x,y)}{\sum_{y'} \exp score(x,y')} = \frac{\exp \sum_i \lambda_i f_i(x,y)}{\sum_{y'} \exp \sum_i \lambda_i f_i(x,y')}$$

    - a powerful tool! (e.g., the log-linear model)

Features based Linear Models: Algorithms

The key is to choose proper weights λs for features

- **the Perceptron algorithm** (covered in this lecture)
- Margin-based models (the Support Vector Machines, SVM)
- Exponential Models:
  - log-linear models, maximum entropy models, logistic models, ...
  - basically, produce a probabilistic model according to score(x, y)

$$p(y|x) = \frac{\exp score(x,y)}{\sum_{y'} \exp score(x,y')} = \frac{\exp \sum_i \lambda_i f_i(x,y)}{\sum_{y'} \exp \sum_i \lambda_i f_i(x,y')}$$

  - a powerful tool! (e.g., the log-linear model)

Outline

The Perceptron Algorithm

- Classic: Rosenblatt (1958)
- Modern: Freund and Schapire (1999)
  - proof for convergence
  - very competitive performance in various classification tasks
- NLP: Michael Collins (2002, 2004, ...)
  - modifications with respect to NLP applications
  - serves as alternative parameter estimation methods for many ML models
  - You SHOULD read at least the 2002 paper (M. Collins, 2002)

The Perceptron Algorithm

- **Inputs**:
  - Training set (x_k, y_k) for $k = 1, 2, ..., n$
  - x_k the data, and y_k the label
- Initialization:
  - $\lambda = [0, 0, 0.....]$, T
- Define:
  - follow Collins: GEN enumerates possible candidate labels ys for data x
  - $z = \arg\max_{y \in GEN(x)} \sum_i \lambda_{f_i(x,y)} f_i(x, y)$
- Loop:
  - For $q = 1, 2, 3..., T$, $k = 1, 2, 3, ..., n$
    compute $z_k = \arg\max_{y \in GEN(x_k)} \sum_i \lambda_{f_i(x,y)} f_i(x_k, y)$ update λs
    - if $z_k \neq y_k$: $\lambda = \lambda + f(x_k, y_k) - f(x_k, z_k)$
- Output:
  - λs

The Perceptron Algorithm

- **Inputs**:
  - Training set (x_k, y_k) for $k = 1, 2, ..., n$
  - x_k the data, and y_k the label
    Here: we treat each word as an instance
- Initialization:
  - $\lambda = [0, 0, 0....]$, T
- Define:
  - follow Collins: GEN enumerates possible candidate labels ys for data x
  - $z = \arg\max_{y \in GEN(x)} \sum_i \lambda_{f_i(x,y)} f_i(x, y)$
- Loop:
  - For $q = 1, 2, 3..., T$, $k = 1, 2, 3, ..., n$
    compute $z_k = \arg\max_{y \in GEN(x_k)} \sum_i \lambda_{f_i(x,y)} f_i(x_k, y)$
    (Key: decode z by ranking over $GEN(x)$)
    update λs
    - if $z_k \neq y_k$: $\lambda = \lambda + f(x_k, y_k) - f(x_k, z_k)$
- Output:
  - λs

A Simple Perceptron Solution for POS Tagging

training data: *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N ...

- in a step during training:
  China/N *Mobile*/N ... *communication*/N **giant**/**??** *in east Asia*
  - word *giant* may have many choices of tags : **N, V, DT, P, ADJ, ...**

A Simple Perceptron Solution for POS Tagging

training data: *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N ...

- in a step during training:
  China/N *Mobile*/N ... *communication*/N **giant/??** *in east Asia*
  - word *giant* may have many choices of tags : **N, V, DT, P, ADJ, ...**
  - for each choice, .e.g, **N**, **ADJ**, we extract m features :
    - $f_1(x, y) = 1$ if current word is *giant* and $y = N$. $\rightarrow f_1(x, y) = 1$
    - $f_{11}(x, y) = 1$ if current word is *giant* and $y = ADJ$. $\rightarrow f_{11}(x, y) = 0$
    - $f_2(x, y) = 1$ if previous word is *the* and $y = N$. $\rightarrow f_2(x, y) = 0$
    - $f_{22}(x, y) = 1$ if previous word is *the* and $y = ADJ$. $\rightarrow f_{22}(x, y) = 0$
    - $f_3(x, y) = 1$ if sufix of current word is *ant* and $y = N$. $\rightarrow f_3(x, y) = 1$
    - $f_{33}(x, y) = 1$ if sufix of current word is *ant* and $y = ADJ$. $\rightarrow f_{33}(x, y) = 0$
    - ...
  - compute score$(giant, N) = \sum_i \lambda_{f_i(giant,N)} f_i(giant, N) = 0.40$, score$(giant, ADJ) = 0.42$, ...
  - choose the largest score$(giant, y)$, e.g., ADJ

A Simple Perceptron Solution for POS Tagging

training data: *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N ...

- in a step during training:
  China/N *Mobile*/N ... *communication*/N **giant**/**??** *in east Asia*
  - word *giant* may have many choices of tags : **N, V, DT, P, ADJ, ...**
  - for each choice, .e.g, **N**, **ADJ**, we extract m features :
    - $f_1(x,y)=1$ if current word is *giant* and $y=N$. $\rightarrow f_1(x,y)=1$
    - $f_{11}(x,y)=1$ if current word is *giant* and $y=ADJ$. $\rightarrow f_{11}(x,y)=0$
    - $f_2(x,y)=1$ if previous word is *the* and $y=N$. $\rightarrow f_2(x,y)=0$
    - $f_{22}(x,y)=1$ if previous word is *the* and $y=ADJ$. $\rightarrow f_{22}(x,y)=0$
    - $f_3(x,y)=1$ if sufix of current word is *ant* and $y=N$. $\rightarrow f_3(x,y)=1$
    - $f_{33}(x,y)=1$ if sufix of current word is *ant* and $y=ADJ$. $\rightarrow$ $f_{33}(x,y)=0$
    - ...
  - compute score$(giant,N)=\sum_i \lambda_{f_i(giant,N)} f_i(giant,N)=0.40$, score$(giant,ADJ)=0.42$, ...
  - choose the largest score$(giant,y)$, e.g., ADJ
- we should at least punish those feature weights that makes us choose ADJ

A Simple Perceptron Solution for POS Tagging

How can we punish those feature weights that makes us choose ADJ?

- the resulting tag is
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/ADJ *in east Asia*
- the gold-standard one
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N

A Simple Perceptron Solution for POS Tagging

How can we punish those feature weights that makes us choose ADJ?

- the resulting tag is
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N **giant/ADJ** *in east Asia*
- the gold-standard one
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N **giant/N** *in*/P *east*/ADJ *Asia*/N

A Simple Perceptron Solution for POS Tagging

How can we punish those feature weights that makes us choose ADJ?

- the resulting tag is
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N **giant/ADJ** *in east Asia*
- the gold-standard one
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N **giant/N** *in*/P *east*/ADJ *Asia*/N
- Which features make us choose the wrong tag ADJ?

A Simple Perceptron Solution for POS Tagging

How can we punish those feature weights that makes us choose ADJ?

- the resulting tag is
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N **giant/ADJ** *in east Asia*
- the gold-standard one
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N **giant/N** *in*/P *east*/ADJ *Asia*/N
- Which features make us choose the wrong tag ADJ?
  - features related to ADJ
  - features related to N

A Simple Perceptron Solution for POS Tagging

How can we punish those feature weights that makes us choose ADJ?

- the resulting tag is
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N **giant/ADJ** *in east Asia*
- the gold-standard one
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N **giant/N** *in*/P *east*/ADJ *Asia*/N
- Which features make us choose the wrong tag ADJ?
  - features related to ADJ
  - features related to N
- **Update** these feature weights accordingly
  - $\lambda^*_{f_1(x,y)} = \lambda_{f_1(x,y)} + 1$
  - $\lambda^*_{f_3(x,y)} = \lambda_{f_3(x,y)} + 1$
  - $\lambda^*_{f_{11}(x,y)} = \lambda_{f_{11}(x,y)} - 1$
  - $\lambda^*_{f_{33}(x,y)} = \lambda_{f_{33}(x,y)} - 1$
  - ...

A Simple Perceptron Solution for POS Tagging

How can we punish those feature weights that makes us choose ADJ?

- the resulting tag is
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N **giant/ADJ** *in east Asia*
- the gold-standard one
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N **giant/N** *in*/P *east*/ADJ *Asia*/N
- Which features make us choose the wrong tag ADJ?
  - features related to ADJ
  - features related to N
- **Update** these feature weights accordingly
  - $\lambda^*_{f_1(x,y)} = \lambda_{f_1(x,y)} + 1$
  - $\lambda^*_{f_3(x,y)} = \lambda_{f_3(x,y)} + 1$
  - $\lambda^*_{f_{11}(x,y)} = \lambda_{f_{11}(x,y)} - 1$
  - $\lambda^*_{f_{33}(x,y)} = \lambda_{f_{33}(x,y)} - 1$
  - ...
- Repeat the process until convergence

Now

What we have:

No.1 Tagger

- An HMM POS Tagger

Feature based Tagger

- A Perceptron POS Tagger
- (A Log-linear POS Tagger)

Now

What we have:

No.1 Tagger

- An HMM POS Tagger
  - NO features at all
  - easy training strategies

Feature based Tagger

- A Perceptron POS Tagger
- (A Log-linear POS Tagger)
  - Rich features
  - need training algorithms

Now

What we have:

No.1 Tagger

- An HMM POS Tagger
  - NO features at all
  - easy training strategies
  - specific decoding: the Viterbi algorithm

Feature based Tagger

- A Perceptron POS Tagger
- (A Log-linear POS Tagger)
  - Rich features
  - need training algorithms
  - How do we decode?

Now

What we have:

No.1 Tagger

- An HMM POS Tagger
  - NO features at all
  - easy training strategies
  - specific decoding: the Viterbi algorithm

Feature based Tagger

- A Perceptron POS Tagger
- (A Log-linear POS Tagger)
  - Rich features
  - need training algorithms
  - can we just do greedy search?

Outline

1. HMM POS Tagger
2. Feature-based Discriminative Models
3. A Perceptron POS Tagger
4. **A Structured Perceptron Tagger**
   - The Viterbi Algorithm
   - Beam Search
5. Tagging with Global Features
6. Neural Sequence Tagger

Are We Done?

- Anything more we can improve?

Are We Done?

- Anything more we can improve?
- Currently, we examine one word by another

Are We Done?

- Anything more we can improve?
- Currently, we examine one word by another
  - individual decisions v.s. a sequence of decisions

Are We Done?

- Anything more we can improve?
- Currently, we examine one word by another
  - individual decisions v.s. a sequence of decisions
- **Local** v.s. **Global**

Are We Done?

- Anything more we can improve?
- Currently, we examine one word by another
  - individual decisions v.s. a sequence of decisions
- **Local** v.s. **Global**
  - Can we utilize the decisions made previously?

Are We Done?

- Anything more we can improve?
- Currently, we examine one word by another
  - individual decisions v.s. a sequence of decisions
- **Local** v.s. **Global**
  - Can we utilize the decisions made previously?
  - Use history decisions as new features

Are We Done?

- Anything more we can improve?
- Currently, we examine one word by another
  - individual decisions v.s. a sequence of decisions
- **Local** v.s. **Global**
  - Can we utilize the decisions made previously?
  - Use history decisions as new features

if the previous word has been tagged as *N* and we want to label the current word w_x as *N* again $\Rightarrow f_{1000000}(w_x, N) = 1$

Are We Done?

- Anything more we can improve?
- Currently, we examine one word by another
  - individual decisions v.s. a sequence of decisions
- **Local** v.s. **Global**
  - Can we utilize the decisions made previously?
  - Use history decisions as new features

if the previous word has been tagged as *N* and we want to label the current word w_x as *N* again $\Rightarrow f_{1000000}(w_x, N) = 1$

- Local models + new/history-based features
  - Perceptron/Log-linear + history-based features

Are We Done?

- Anything more we can improve?
- Currently, we examine one word by another
  - individual decisions v.s. a sequence of decisions
- **Local** v.s. **Global**
  - Can we utilize the decisions made previously?
  - Use history decisions as new features

if the previous word has been tagged as *N* and we want to label the current word w_x as *N* again $\Rightarrow f_{1000000}(w_x, N) = 1$

- Local models + new/history-based features
  - Perceptron/Log-linear + history-based features
- Need specific decoding algorithms?

A Variant: The Structured Perceptron Algorithm

- **Inputs**:
  - Training set $S : (x_k, y_k)$ for $k = 1, 2, ..., n$, belonging to $|S|$ sentences
  - x_k the data, and y_k the label,
- Initialization:
  - $\lambda = [0, 0, 0.....]$, T
- Define:
  - GEN enumerates possible candidate label ys for data x
  - $score_y = \sum_i \lambda_{f_i(x,y)} f_i(x, y)$: compute the score for a pair of x and y
- Loop:
  - For $q = 1, 2, 3..., T$, each sentence $s \in S$
    for each word $x \in s$, $y \in$ GEN(x):
    - compute $score_y = \sum_i \lambda_{f_i(x,y)} f_i(x, y)$

    Decode the **best sequence** for sentence s
    Update λs regarding sentence s
- Output:
  - λs

A Variant: The Structured Perceptron Algorithm

- **Inputs**:
  - Training set $S : (x_k, y_k)$ for $k = 1, 2, ..., n$, belonging to $|S|$ sentences
  - x_k the data, and y_k the label, treat one sentence as one instance?
- Initialization:
  - $\lambda = [0, 0, 0.....]$, T
- Define:
  - GEN enumerates possible candidate label ys for data x
  - $score_y = \sum_i \lambda_{f_i(x,y)} f_i(x, y)$: compute the score for a pair of x and y
- Loop:
  - For $q = 1, 2, 3..., T$, each sentence $s \in S$
    for each word $x \in s$, $y \in$ GEN(x):
    - compute $score_y = \sum_i \lambda_{f_i(x,y)} f_i(x, y)$

    Decode the **best sequence** for sentence s
    Update λs regarding sentence s
- Output:
  - λs

A Variant: The Structured Perceptron Algorithm

- **Inputs**:
  - Training set $S : (x_k, y_k)$ for $k = 1, 2, ..., n$, belonging to $|S|$ sentences
  - x_k the data, and y_k the label, treat one sentence as one instance?
- Initialization:
  - $\lambda = [0, 0, 0....]$, T
- Define:
  - GEN enumerates possible candidate label ys for data x
  - $score_y = \sum_i \lambda_{f_i(x,y)} f_i(x, y)$: compute the score for a pair of x and y
- Loop:
  - For $q = 1, 2, 3..., T$, each sentence $s \in S$
    for each word $x \in s$, $y \in$ GEN(x):
    - compute $score_y = \sum_i \lambda_{f_i(x,y)} f_i(x, y)$ $\rightarrow$ **a lattice ($|s| \times |\mathbf{y}|$)**

    Decode the **best sequence** for sentence s
    Update λs regarding sentence s
- Output:
  - λs

A Variant: The Structured Perceptron Algorithm

- **Inputs**:
  - Training set $S : (x_k, y_k)$ for $k = 1, 2, ..., n$, belonging to $|S|$ sentences
  - x_k the data, and y_k the label, treat one sentence as one instance?
- Initialization:
  - $\lambda = [0, 0, 0.....]$, T
- Define:
  - GEN enumerates possible candidate label ys for data x
  - $score_y = \sum_i \lambda_{f_i(x,y)} f_i(x, y)$: compute the score for a pair of x and y
- Loop:
  - For $q = 1, 2, 3..., T$, each sentence $s \in S$
    for each word $x \in s$, $y \in$ GEN(x):
    - compute $score_y = \sum_i \lambda_{f_i(x,y)} f_i(x, y)$ $\rightarrow$ **a lattice ($|s| \times |$y$|$)**

    Decode the **best sequence** for sentence s **with the Viterbi Algorithm**
    Update λs regarding sentence s **by comparing**
    the currently best y sequence with its gold-standard y* sequence
- Output:
  - λs

Perceptron v.s. Structured Perceptron

Perceptron

- only use all words in the sentence as features
- use greedy search as the decoder

- a local solution

Structured Perceptron

- besides local features, can also take previous decisions, e.g., y_{k-2}, y_{k-1}, as features
- use the Viterbi Algorithm or others as the decoder

- a solution with more global-views/history-views

A Structured Perceptron Solution for POS Tagging

training data: *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N

- at time t during training
  - each word $x = \{$China, Mobile, ..., Asia$\}$ in the sentence, try every every possible tag: **N, V, DT, P, ADJ, ...**

A Structured Perceptron Solution for POS Tagging

training data: *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N

- at time t during training
  - each word $x = \{$China, Mobile, ..., Asia$\}$ in the sentence, try every every possible tag: **N, V, DT, P, ADJ, ...**
  - for each choice, .e.g, N, we extract m features :
    - $f_1(x,y) = 1$ if current word is *China* and $y = N$. $\rightarrow f_1(x,y) = 1$
    - $f_{11}(x,y) = 1$ if current word is *China* and $y = ADJ$. $\rightarrow f_{11}(x,y) = 0$
    - $f_2(x,y) = 1$ if previous word is $<$*S*$>$ and $y = N$. $\rightarrow f_2(x,y) = 0$
    - $f_{22}(x,y) = 1$ if previous word is $<$*S*$>$ and $y = ADJ$. $\rightarrow f_{22}(x,y) = 0$
    - $f_3(x,y) = 1$ if prefix of current word is *Chi* and $y = N$. $\rightarrow f_3(x,y) = 1$
    - $f_{33}(x,y) = 1$ if prefix of current word is *Chi* and $y = ADJ$. $\rightarrow f_{33}(x,y) = 0$
    - ...
  - compute and keep score(x, N), score(x, ADJ), score(x, DT), ...

A Structured Perceptron Solution for POS Tagging

training data: *China*/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N

- at time t during training
  - each word $x = \{$China, Mobile, ..., Asia$\}$ in the sentence, try every every possible tag: **N, V, DT, P, ADJ, ...**
  - for each choice, .e.g, N, we extract m features :
    - $f_1(x, y) = 1$ if current word is *China* and $y = N$. $\rightarrow f_1(x, y) = 1$
    - $f_{11}(x, y) = 1$ if current word is *China* and $y = ADJ$. $\rightarrow f_{11}(x, y) = 0$
    - $f_2(x, y) = 1$ if previous word is $<S>$ and $y = N$. $\rightarrow f_2(x, y) = 0$
    - $f_{22}(x, y) = 1$ if previous word is $<S>$ and $y = ADJ$. $\rightarrow f_{22}(x, y) = 0$
    - $f_3(x, y) = 1$ if prefix of current word is *Chi* and $y = N$. $\rightarrow f_3(x, y) = 1$
    - $f_{33}(x, y) = 1$ if prefix of current word is *Chi* and $y = ADJ$. $\rightarrow f_{33}(x, y) = 0$
    - ...
  - compute and keep score(x, N), score(x, ADJ), score(x, DT), ...
- Build **a lattice ($|s| \times |\mathbf{y}|$)** for this sentence
- Decode **the best sequence** for this sentence with **the Viterbi Algorithm**

Structured Perceptron for POS Tagging (A simple case)

- the resulting sequence is
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/ADJ *in*/DT *east*/ADJ *Asia*/N
- the gold-standard one
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N

Structured Perceptron for POS Tagging (A simple case)

- the resulting sequence is
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/ADJ *in*/DT *east*/ADJ *Asia*/N
- the gold-standard one
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N
- we compare the two sequences, and find the differences

Structured Perceptron for POS Tagging (A simple case)

- the resulting sequence is
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N <u>*giant*/ADJ *in*/DT</u> *east*/ADJ *Asia*/N
- the gold-standard one
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N <u>*giant*/N *in*/P</u> *east*/ADJ *Asia*/N
- we compare the two sequences, and find the differences
- we **update** the features related to the correct/wrong predictions
- for example, we should do something regarding <u>*giant*/ADJ *in*/DT</u>
  - $\lambda^*_{f_1(x,y)} = \lambda_{f_1(x,y)} + 1$
  - $\lambda^*_{f_3(x,y)} = \lambda_{f_3(x,y)} + 1$
  - $\lambda^*_{f_{11}(x,y)} = \lambda_{f_{11}(x,y)} - 1$
  - $\lambda^*_{f_{33}(x,y)} = \lambda_{f_{33}(x,y)} - 1$
  - ...

Structured Perceptron for POS Tagging (A simple case)

- the resulting sequence is
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/ADJ *in*/DT *east*/ADJ *Asia*/N
- the gold-standard one
  China/N *Mobile*/N *is*/V *a*/DT *communication*/N *giant*/N *in*/P *east*/ADJ *Asia*/N
- we compare the two sequences, and find the differences
- we **update** the features related to the correct/wrong predictions
- for example, we should do something regarding *giant*/ADJ *in*/DT
  - $\lambda^*_{f_1(x,y)} = \lambda_{f_1(x,y)} + 1$
  - $\lambda^*_{f_3(x,y)} = \lambda_{f_3(x,y)} + 1$
  - $\lambda^*_{f_{11}(x,y)} = \lambda_{f_{11}(x,y)} - 1$
  - $\lambda^*_{f_{33}(x,y)} = \lambda_{f_{33}(x,y)} - 1$
  - ...
- update the parameters **in a sentence level**

Outline

1. HMM POS Tagger
2. Feature-based Discriminative Models
3. A Perceptron POS Tagger
4. **A Structured Perceptron Tagger**
   - The Viterbi Algorithm
   - Beam Search
5. Tagging with Global Features
6. Neural Sequence Tagger

A Bit Complex: Why We Need the Viterbi Algorithm

If we include history-based features like

- $f_{100}(x, y) = 1$ if previous tag is *N* and $y = N$. $\rightarrow f_{100}(giant, N) = 1$
- $f_{101}(x, y) = 1$ if previous two tags are *DT_N* and $y = N$. $\rightarrow f_{101}(giant, N) = 1$
- ...
- we **can NOT directly/individually** compute score$(giant, N)$, score$(giant, ADJ)$, ...

A Bit Complex: Why We Need the Viterbi Algorithm

If we include history-based features like

- $f_{100}(x, y) = 1$ if previous tag is *N* and $y = N$. $\rightarrow f_{100}(giant, N) = 1$
- $f_{101}(x, y) = 1$ if previous two tags are *DT_N* and $y = N$. $\rightarrow f_{101}(giant, N) = 1$
- ...
- we **can NOT directly/individually** compute score$(giant, N)$, score$(giant, ADJ)$, ...
- we need to decode the **currently best** tag sequence for the whole sentence using Dynamic Programming

A Bit Complex: Why We Need the Viterbi Algorithm

If we include history-based features like

- $f_{100}(x, y) = 1$ if previous tag is *N* and $y = N$. $\rightarrow f_{100}(giant, N) = 1$
- $f_{101}(x, y) = 1$ if previous two tags are *DT_N* and $y = N$. $\rightarrow$ $f_{101}(giant, N) = 1$
- ...
- we **can NOT directly/individually** compute score$(giant, N)$, score$(giant, ADJ)$, ...
- we need to decode the **currently best** tag sequence for the whole sentence using Dynamic Programming $\rightarrow$ **the Viterbi Algorithm**
-

$$\arg \max_{t_{[1:n]} \in GEN'(s)} \sum_{w \in s, y \in t_{[1:n]}} \sum_i \lambda_{f_i(\text{history}(w), y)} f_i(\text{history}(w), y)$$

Decoding: the Viterbi Algorithm

- for a sentence s of length n
- define the score of tag sequence $t_1, t_2, ...t_j$:
  score$(t_1, t_2, ...t_j) = \sum_{w \in s} \sum_i \lambda_{f'_i(w, t_{w-2}, t_{w-1}, t_w)} f'_i(w, t_{w-2}, t_{w-1}, t_w)$
- define the dynamic programming table
  $\pi(j, u, v) =$ maximum probability of a tag sequence ending with tags u, v at position j
- so,

$$\pi(j, u, v) = \max_{<t_1, t_2, ...t_{j-2}>} \text{score}(t_1, t_2, ...t_{j-2}, u, v)$$

- Recursively:
  start with $\pi(0, \text{START}, \text{START}) = 0$
  for any $j \in 1, 2, ..., n$, for possible u and v:

$$\pi(j, u, v) = \max_q (\pi(j-1, q, u) + \sum_i \lambda_{f'_i(word_v, q, u, v)} f'_i(word_v, q, u, v))$$

- the Viterbi Algorithm with Backpointers $\rightarrow$ the optimal sequence!

Outline

An Alternative: Beam Search

Input: an input sentence x of length n, a trained model $score(*)$, and a predefined beam size k

Let H_i store the current hypotheses (or, possible results) at position $i \in \{1, 2, ..., n\}$:

- let C be a temporal storage
- for each hypothesis (or, possible result) $y^H_{1:i-1}$ in H_{i-1} at position $i-1$
  - try every possible y^h_i, and form a new tag sequence, $y^H_{1:i-1}, y^h_i$, and store it with its score in C
- Choose the k-best sequences in C to form the H_i

Output: the best-scored sequence in H_i

An Alternative: Beam Search

Input: an input sentence x of length n, a trained model $score(*)$, and a predefined beam size k

Let H_i store the current hypotheses (or, possible results) at position $i \in \{1, 2, ..., n\}$:

- let C be a temporal storage
- for each hypothesis (or, possible result) $y^H_{1:i-1}$ in H_{i-1} at position $i-1$
  - try every possible y^h_i, and form a new tag sequence, $y^H_{1:i-1}, y^h_i$, and store it with its score in C
- Choose the k-best sequences in C to form the H_i

Output: the best-scored sequence in H_i

- Easy to implement, Effective and Efficient in most of time
- Generally no guarantee for global optimal :-(
- Runtime is $O(n^2 k |\mathbf{y}|)$, space is $O(n^2 k)$

More about Structured Perceptron

- Voted Perceptron (Collins 2002)
- Averaged Perceptron (Collins 2002)
- Early Update (Collins and Roak 2004)

Questions

- can this model take features like:
  how many times we see a verb in this sentence ?
  Is there a verb appearing in this sentence?

Perceptron

- Rosenblatt, 1958
- Freund and Schapire, 1999
- Collins, 2002
- Collins and Roak, 2004
- ...

You see `Perceptron` in neural times as well

- Transition system
- Multiple Layer Perceptron (MLP)!

Outline

Just Another View: Global and Local Features

- Local features are indicator functions, e.g.,

$$f_{101}(w_i, t_i) = \begin{cases} 1 & \text{if current word } w_i \text{ ends in } \textbf{ing} \text{ and } t_i\text{=VBG} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{109}(w_i, t_{[i-1,i]}) = \begin{cases} 1 & \text{if } t_{i-1}\text{=ADJ and } t_i\text{=VBG} \\ 0 & \text{otherwise} \end{cases}$$

- Then, global features can be simply counts:
  - $F_{101}(w_{[1:n]}, t_{[1:n]})$ is the number of times that a word ending in **ing** is tagged as VBG in $(w_{[1:n]}, t_{[1:n]})$:
    $F_{101}(w_{[1:n]}, t_{[1:n]}) = \sum_{i=1}^{n} f_{101}(w_i, t_i)$
  - $F_{109}(w_{[1:n]}, t_{[1:n]})$ is the number of times that a word is tagged as VBG and its previous neighbor tagged as ADJ in $(w_{[1:n]}, t_{[1:n]})$:
    $F_{109}(w_{[1:n]}, t_{[1:n]}) = \sum_{i=1}^{n} f_{109}(w_i, t_{[i-1,i]})$

Plug into a Log-linear Model

- Look in a sentence level: $p(Y|X)$ the probability of one sentence X labeled with tag sequence Y:

$$
\begin{aligned}
Y^* = \arg\max p(Y|X) &= \arg\max \frac{\exp(\boldsymbol{\lambda} \cdot \boldsymbol{F}(X,Y))}{\sum_{Y'} \exp(\boldsymbol{\lambda} \cdot \boldsymbol{F}(X,Y'))} \\
&= \arg\max \frac{\exp(\sum_{m=1}^{M} \lambda_m F_m(X,Y))}{\sum_{Y'} \exp(\sum_{m=1}^{M} \lambda_m F_m(X,Y'))} \\
&= \arg\max \frac{1}{Z(X)} \exp(\sum_{m=1}^{M} \lambda_m F_m(X,Y))
\end{aligned}
$$

  - slightly simply with: $Z(X) = \sum_{Y'} \exp(\sum_{m=1}^{M} \lambda_m F_m(X,Y'))$
  - $F_m(X,Y)$ could be history-based or sentence level global features

Plug into a Log-linear Model

- Look in a sentence level: $p(Y|X)$ the probability of one sentence X labeled with tag sequence Y:

$$Y^* = \arg\max p(Y|X) = \arg\max \frac{\exp(\boldsymbol{\lambda} \cdot \boldsymbol{F}(X,Y))}{\sum_{Y'} \exp(\boldsymbol{\lambda} \cdot \boldsymbol{F}(X,Y'))}$$

$$= \arg\max \frac{\exp(\sum_{m=1}^{M} \lambda_m F_m(X,Y))}{\sum_{Y'} \exp(\sum_{m=1}^{M} \lambda_m F_m(X,Y'))}$$

$$= \arg\max \frac{1}{Z(X)} \exp(\sum_{m=1}^{M} \lambda_m F_m(X,Y))$$

  - slightly simply with: $Z(X) = \sum_{Y'} \exp(\sum_{m=1}^{M} \lambda_m F_m(X,Y'))$
  - $F_m(X,Y)$ could be history-based or sentence level global features
  - training objective: $-\sum \log p(Y_i|X_i) = \sum -\boldsymbol{\lambda} \cdot \boldsymbol{F}(X_i,Y_i) + \log Z(X_i)$

Plug into a Log-linear Model

- Look in a sentence level: $p(Y|X)$ the probability of one sentence X labeled with tag sequence Y:

$$Y^* = \arg\max p(Y|X) = \arg\max \frac{\exp(\boldsymbol{\lambda} \cdot \boldsymbol{F}(X,Y))}{\sum_{Y'} \exp(\boldsymbol{\lambda} \cdot \boldsymbol{F}(X,Y'))}$$

$$= \arg\max \frac{\exp(\sum_{m=1}^{M} \lambda_m F_m(X,Y))}{\sum_{Y'} \exp(\sum_{m=1}^{M} \lambda_m F_m(X,Y'))}$$

$$= \arg\max \frac{1}{Z(X)} \exp(\sum_{m=1}^{M} \lambda_m F_m(X,Y))$$

  - slightly simply with: $Z(X) = \sum_{Y'} \exp(\sum_{m=1}^{M} \lambda_m F_m(X,Y'))$
  - $F_m(X,Y)$ could be history-based or sentence level global features
  - training objective: $-\sum \log p(Y_i|X_i) = \sum -\boldsymbol{\lambda} \cdot \boldsymbol{F}(X_i,Y_i) + \log Z(X_i)$
  - going over all possible Ys is horrible! $\rightarrow$ $Z(X)$!!!

Conditional Random Fields (Lafferty et al. (2001))

This is a (linear chain) Conditional Random Fields (CRF) model.

- one of the most influential models in statistical learning for structured predictions, especially sequence tagging.
- usually in the following form, with Y as various target structures.

$$Y^* = \arg\max \frac{1}{Z(X)} \exp(\sum_{m=1}^{M} \lambda_m F_m(X, Y))$$

Conditional Random Fields (Lafferty et al. (2001))

This is a (linear chain) Conditional Random Fields (CRF) model.

- one of the most influential models in statistical learning for structured predictions, especially sequence tagging.
- usually in the following form, with Y as various target structures.

$$Y^* = \arg\max \frac{1}{Z(X)} \exp(\sum_{m=1}^{M} \lambda_m F_m(X, Y))$$

- trained with MLE

$$-\sum \log p(Y_i|X_i) = \sum -\boldsymbol{\lambda} \cdot \boldsymbol{F}(X_i, Y_i) + \log Z(X_i)$$

Conditional Random Fields (Lafferty et al. (2001))

This is a (linear chain) Conditional Random Fields (CRF) model.

- one of the most influential models in statistical learning for structured predictions, especially sequence tagging.
- usually in the following form, with Y as various target structures.

$$Y^* = \arg\max \frac{1}{Z(X)} \exp(\sum_{m=1}^{M} \lambda_m F_m(X, Y))$$

- trained with MLE

$$-\sum \log p(Y_i|X_i) = \sum -\boldsymbol{\lambda} \cdot \boldsymbol{F}(X_i, Y_i) + \log Z(X_i)$$

- use SGD if we can calculate and differentiate the $F(*)$ and $Z(*)$

Conditional Random Fields (Lafferty et al. (2001))

This is a (linear chain) Conditional Random Fields (CRF) model.

- one of the most influential models in statistical learning for structured predictions, especially sequence tagging.
- usually in the following form, with Y as various target structures.

$$Y^* = \arg\max \frac{1}{Z(X)} \exp(\sum_{m=1}^{M} \lambda_m F_m(X, Y))$$

- trained with MLE

$$-\sum \log p(Y_i|X_i) = \sum -\boldsymbol{\lambda} \cdot \boldsymbol{F}(X_i, Y_i) + \log Z(X_i)$$

- use SGD if we can calculate and differentiate the $F(*)$ and $Z(*)$
- How to decode? the Forward Algorithm! very similar to the Viterbi algorithm

Now

What we have:

Sequential Tagger

- An HMM POS Tagger
- A Structured Perceptron POS Tagger
- A CRF POS Tagger

Local Tagger

- A Perceptron POS Tagger
- (A Log-linear POS Tagger)

Now

What we have:

Sequential Tagger

- An HMM POS Tagger
- A Structured Perceptron POS Tagger
- A CRF POS Tagger

Local Tagger

- A Perceptron POS Tagger
- (A Log-linear POS Tagger)

- various learning tricks: counting, simple plus, SGD
- various decoding tricks: greedy, Viterbi, beam search
- remember to regularize!

Outline

In Neural Times

Traditionally:

- Feature engineering
- Language issues
- Out of vocabulary (OOV)
- Local/Global
- Label bias

In Neural Times

Traditionally: $\rightarrow$ SOTA: Conditional Random Field (CRF)

- Feature engineering
- Language issues
- Out of vocabulary (OOV)
- Local/Global
- Label bias

In Neural Times

Traditionally: → SOTA: Conditional Random Field (CRF)

- Feature engineering
- Language issues
- Out of vocabulary (OOV)
- Local/Global
- Label bias

Neural Times:

- No feature engineering
- No language barriers
  - Continuous representations
  - Letter/Character levels
  - glyph level (stroke)
- Various NN architectures to capture local/long context, both forward and backward
  - CNN, RNN, LSTM, BLSTM, BLSTM-CNN

In Neural Times

Traditionally: $\rightarrow$ SOTA: Conditional Random Field (CRF)

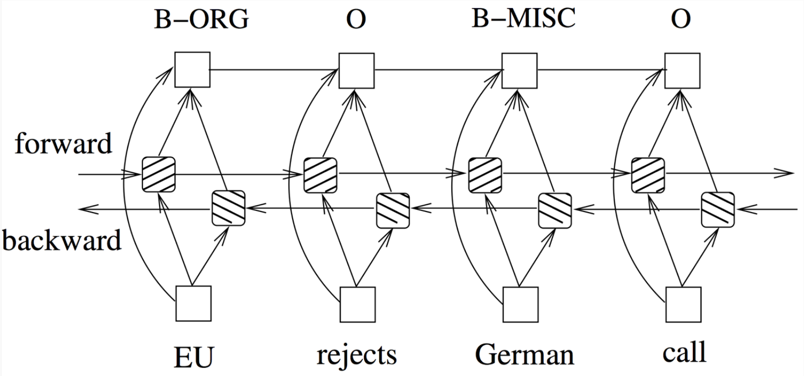
- Feature engineering
- Language issues
- Out of vocabulary (OOV)
- Local/Global
- Label bias

Neural Times:

- No feature engineering
- No language barriers
  - Continuous representations
  - Letter/Character levels
  - glyph level (stroke)
- Various NN architectures to capture local/long context, both forward and backward
  - CNN, RNN, LSTM, BLSTM, BLSTM-CNN
- CRF with Viterbi to find the best sequence

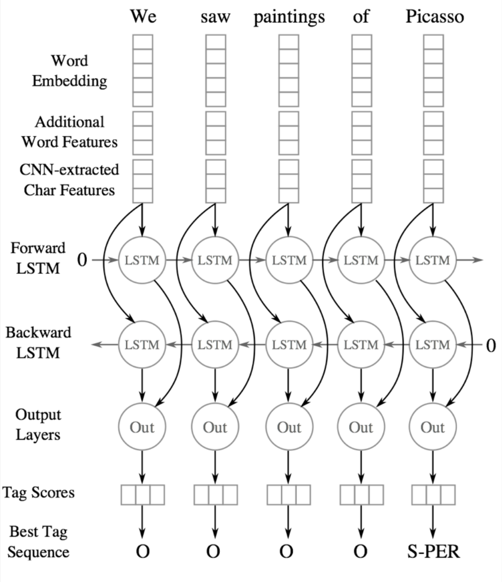
BSLTM for NER

Vanilla BLSTM [Huang et al., 2015]



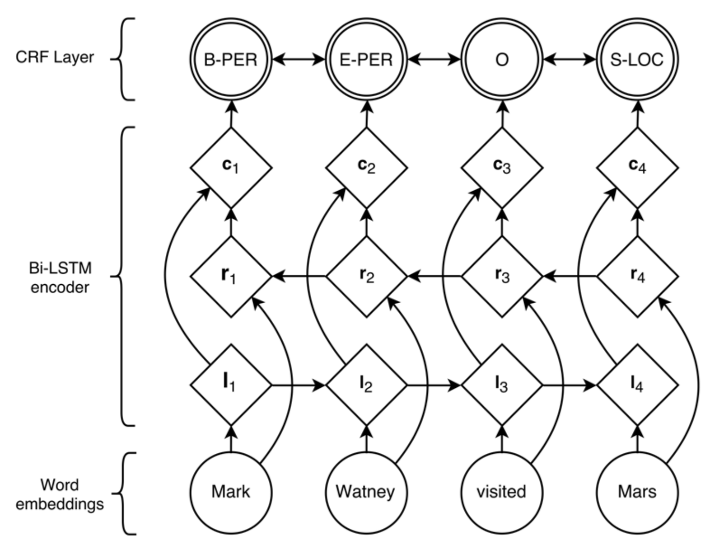
BSLTM for NER

BLSTM with CNN [Chiu and Nicols., 2016]



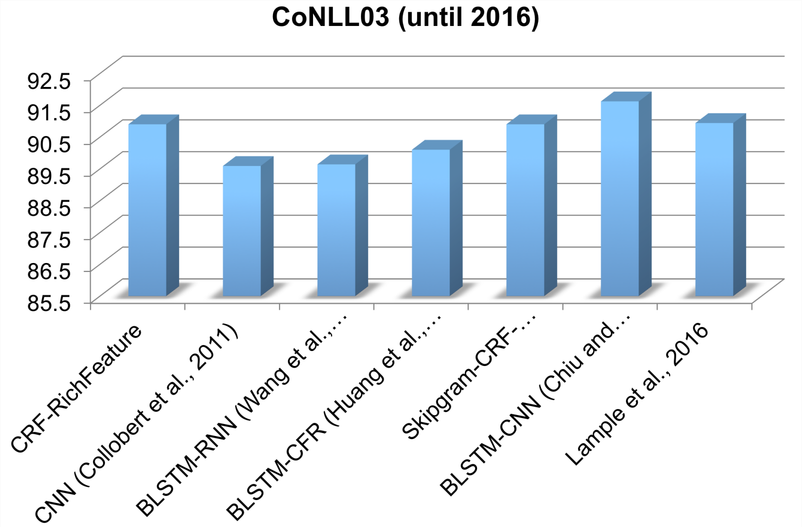
BSLTM for NER

BLSTM with CRF (conditional random field) [Lample et al., 2016]



NER on CoNLL 03

NER performance on CoNLL03 until 2016



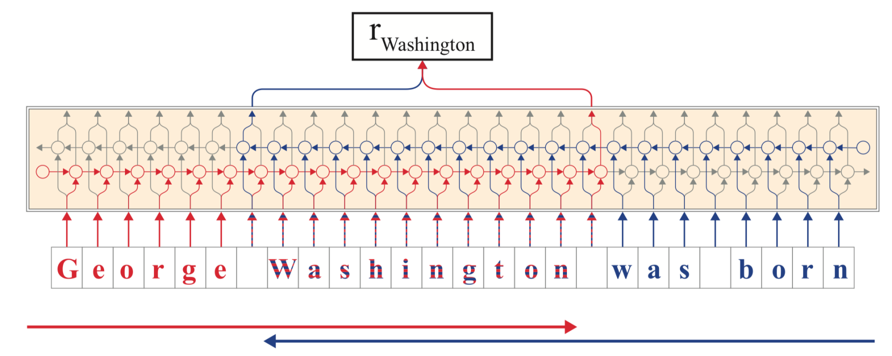
Modeling the Context

Context: LSTM, BLSTM, CRF, ...

33	CRF + AutoEncoder	91.87	×	Evaluating the Utility of Hand-crafted Features in Sequence Labelling	2018
34	PRISM	91.8	×	A Prism Module for Semantic Disentanglement in Name Entity Recognition	2019
35	GraphIE (GCN+BiLSTM)	91.74	×		2019
36	Bi-LSTM-CRF + Lexical Features	91.73	×	Robust Lexical Features for Improved Neural Network Named-Entity Recognition	2018
37	IntNet + BiLSTM-CRF	91.64	×	Learning Better Internal Structure of Words for Sequence Labeling	2018
38	Yang et al. ([2017a])	91.62	×	Neural Reranking for Named Entity Recognition	2017
39	Bi-LSTM-CNN	91.62	×	Named Entity Recognition with Bidirectional LSTM-CNNs	2015
40	S-LSTM	91.57	×	Sentence-State LSTM for Text Representation	2018
41	LSTM with dynamic skip	91.56	×	Long Short-Term Memory with Dynamic Skip Connections	2018
42	Adversarial Bi-LSTM	91.56	×	Robust Multilingual Part-of-Speech Tagging via Adversarial Training	2017
43	HSCRF	91.38	×	Hybrid semi-Markov CRF for Neural Sequence Labeling	2018
44	IXA pipes	91.36	×	Robust Multilingual Named Entity Recognition with Shallow Semi-Supervised Features	2017
45	NCRF++	91.35	×	NCRF++: An Open-source Neural Sequence Labeling Toolkit	2018
46	Yang et al.	91.26	×	Transfer Learning for Sequence Tagging with Hierarchical Recurrent Networks	2017
47	LM-LSTM-CRF	91.24	×	Empower Sequence Labeling with Task-Aware Neural Language Model	2017
48	Bi-LSTM-CNN-CRF	91.22	×	A Deep Neural Network Model for the Task of Named	2018

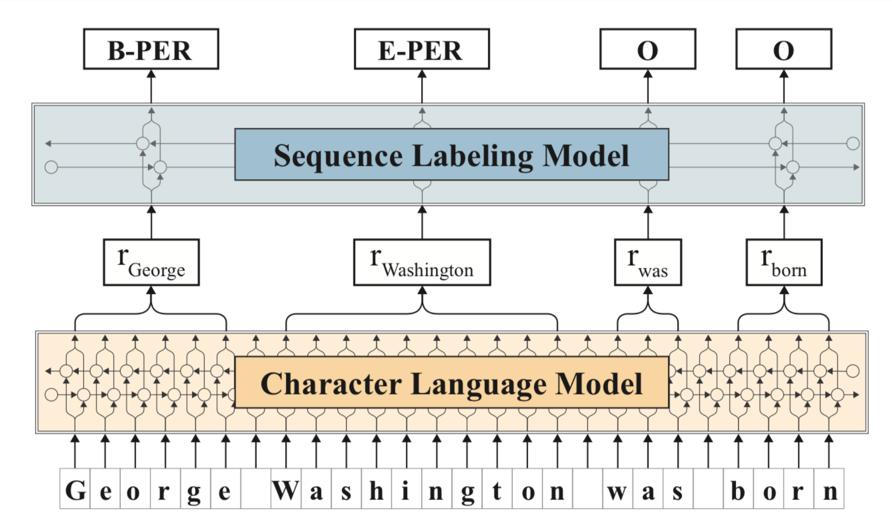
Contextualized (String) Embedding

Flair: contextual string embeddings



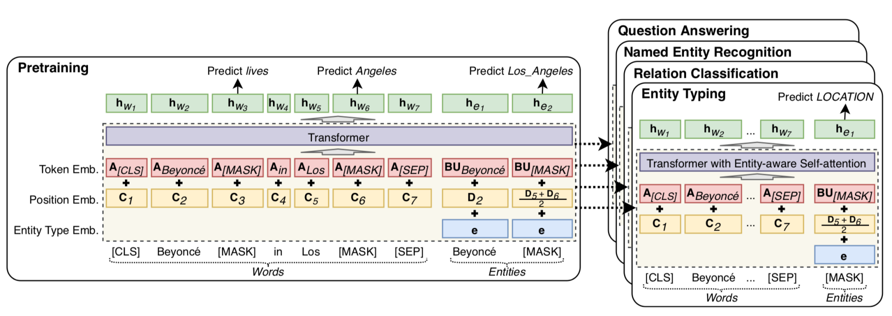
Contextualized (String) Embedding

Flair: contextual string embeddings



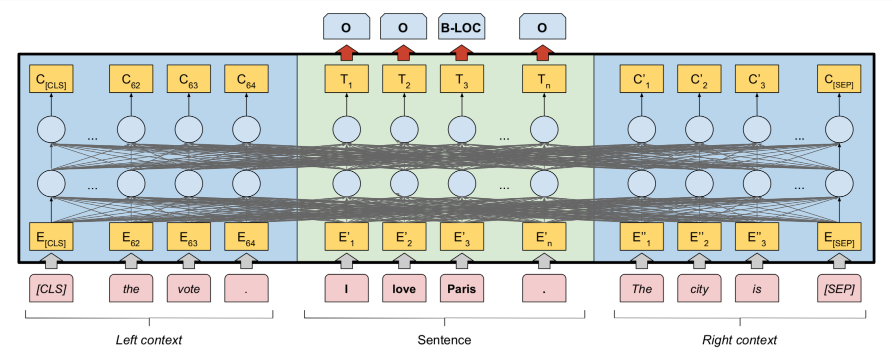
Language Understanding with Knowledge-based Embeddings

LUKE: transformer (BERT), masked entity predition, entity-aware self-attention, ...



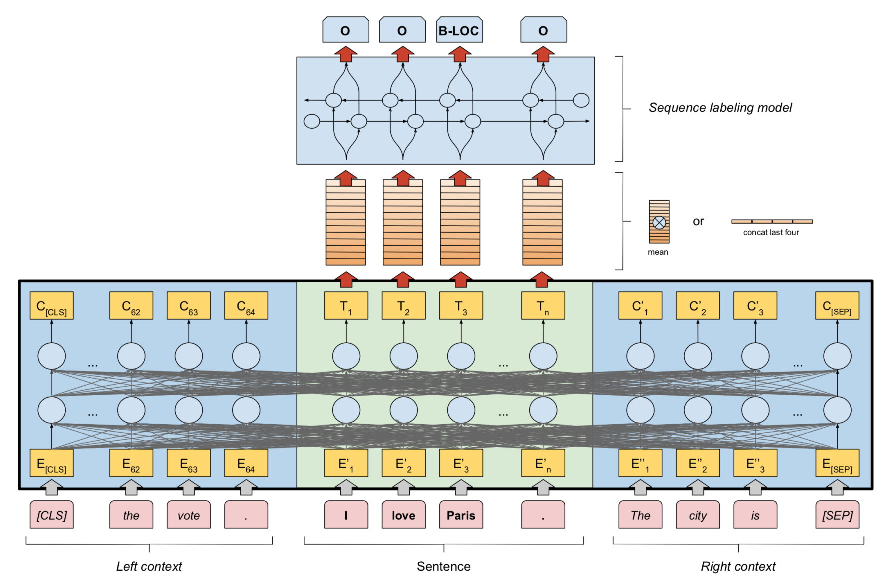
Using Document Level Features!

FLERT: Document level features for NER



Using Document Level Features!

FLERT: Document level features for NER



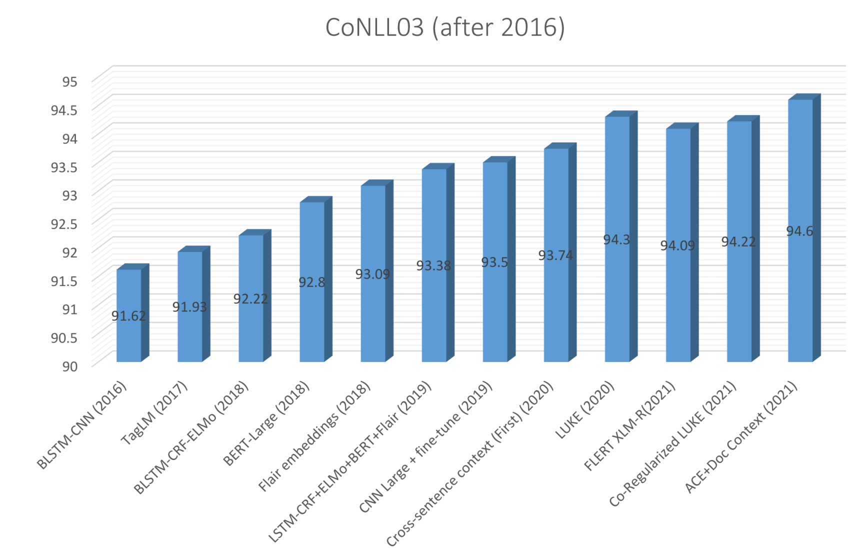
Modeling the Context

Context: transformer, contextualized, Bert, attention, ...

1	ACE + document-context	94.6	×	Automated Concatenation of Embeddings for Structured Prediction			2021	LSTM Transformer
2	Co-regularized LUKE	94.22	×	Learning from Noisy Labels for Entity-Centric Information Extraction			2021	knowledge distillati
3	ASP+T5-3B	94.1	×	Autoregressive Structured Prediction with Language Models			2022	
4	FLERT XLM-R	94.09	×	FLERT: Document-Level Features for Named Entity Recognition			2020	Transformer
5	PL-Marker	94.0	×	Packed Levitated Marker for Entity and Relation Extraction			2021	
6	LUKE	93.91	×	LUKE: Deep Contextualized Entity Representations with Entity-aware Self-attention			2020	Transformer
7	CL-KL	93.85	×	Improving Named Entity Recognition by External Context Retrieving and Cooperative Learning			2021	Transformer
8	XLNet-GCN	93.82	×	Named entity recognition architecture combining contextual and global features			2020	
9	ASP+flan-T5-large	93.8	×	Autoregressive Structured Prediction with Language Models			2022	
10	InferNER	93.76	×	InferNER: an attentive model leveraging the sentence-level information for Named Entity Recognition in Microblogs			2021	LSTM
11	Cross-sentence context (First)	93.74	×	Exploring Cross-sentence Contexts for Named Entity Recognition with BERT			2020	Transformer
12	Baseline + BS	93.65	×	Boundary Smoothing for Named Entity Recognition			2022	
13	ACE	93.64	×	Automated Concatenation of Embeddings for Structured Prediction			2020	

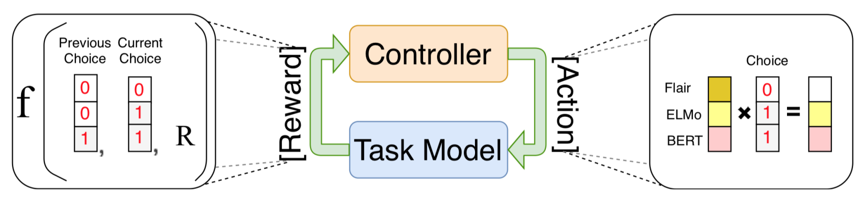
NER on CoNLL03

NER performance on CoNLL03 after 2016 (until 2022)



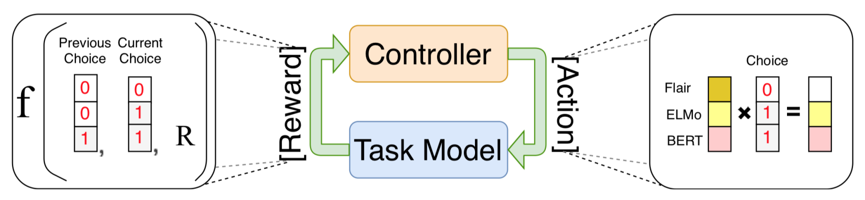
Beyond Contextual Embeddings

Maybe we need a bit more machine/deep learning...



Beyond Contextual Embeddings

Maybe we need a bit more machine/deep learning...
Automated Concatenation of Embeddings $\Rightarrow$
Neural Architecture Search



The Key

`The Context.`

Readings

SLP-3 Chapter 17, Speech and Language Processing (**SLP**)

1999 Large Margin Classification using the Perceptron Algorithm, Machine Learning, 1999

2001 John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proc. of ICML, 2001.

2002 Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. Michael Collins, EMNLP, 2002

2004 Incremental parsing with the Perceptron algorithm. Michael Collins and Brian Roark, ACL, 2004

2016 Methods and theories for large-scale structured prediction. Xu Sun and Yansong Feng, EMNLP Tutorial, 2016

Reference

- Alexandre Passos, Vineet Kumar, Andrew McCallum, Lexicon Infused Phrase Embeddings for Named Entity Resolution, In Proceedings of the Eighteenth Conference on Computational Language Learning, pages 78-86, Baltimore, Maryland USA, June 26-27 2014.
- Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov and Michael Collin, Globally Normalized Transition-Based Neural Networks, Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 2442-2452, Berlin, Germany, August 7-12, 2016
- Jason P.C. Chiu and Eric Nichols, Named Entity Recognition with Bidirectional LSTM-CNNs, In Transactions of the Association for Computational Linguistics, vol. 4, pp. 357-370, 2016
- Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. The Journal of Machine Learning Research, 12:2493-2537.

Reference

- Wang Ling, Tiago Lus, Lus Marujo, Ramon Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, Isabel Trancoso, Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation, In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1520-1530, Lisbon, Portugal, 17-21 September 2015.
- Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidi- rectional LSTM-CRF models for sequence tagging. CoRR,abs/1508.01991.
- Guillaume Lample,Miguel Ballesteros,Sandeep Subramanian,Kazuya KawakamiandChris Dyer, Neural Architectures for Named Entity Recognition, NAACL 2016, 260-270
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, NAACL 2019

Reference

- Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, Michael Auli, Cloze-driven Pretraining of Self-attention Networks, IJCNLP 2019
- Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda andYuji Matsumoto, LUKE: Deep Contextualized Entity Representations with Entity-aware Self-attention, EMNLP 2020