

# Introduction to NLP

---

Session 7

# Sequential Tagging Problems

- Many problems in NLP require text to be tagged:
  - Part-of-Speech Tagging (POS): Find lexical terms in a sentence.
    - Input: 'red', 'flower'
    - Output: 'adjective', 'noun'
  - Named Entity Recognition (NER): Find 'Entities' in a sentence.
    - Input: 'Bill', 'Gates', 'founded', 'Microsoft', 'decades', 'ago'
    - Output: 'Person', 'Person', '-', 'Company', 'O', 'O'

# Sequential Labelling Problems Difficulties

- Example: 'Bill', 'Gates', 'founded', 'Microsoft', 'decades', 'ago'
- We would like to know that 'Bill Gates' is a person, not consider 'Bill' and 'Gates' as two different people
- **Inside-Outside-Beginning Tagging (IOB Tagging)** uses B (Beginning) and I (Inside) tags to change the class label of a word to allow multiple words to belong to a single concept
  - Input: 'Bill', 'Gates', 'founded', 'Microsoft', 'decades', 'ago'
  - Output: 'B-Per', 'I-Per', 'O', 'B-COMP', 'O', 'O'
  - Input: Angelina Jolie was born in Los Angeles.
  - Output: 'B-PER', 'I-PER', 'O', 'O', 'O', 'B-LOC', 'I-LOC'

# Sequential Labelling is Hard

<https://huggingface.co/dbmdz/bert-large-cased-finetuned-conll03-english>

The screenshot shows the Hugging Face interface for the model `dbmdz/bert-large-cased-finetuned-conll03-english`. The header includes the Hugging Face logo, a search bar, and navigation links for Models, Datasets, Resources, Solutions, Pricing, Log In, and Sign Up. The model name is displayed with a like count of 0. Below the name are tags for Token Classification, PyTorch, TensorFlow, JAX, Rust, Transformers, and bert. The Model card tab is selected, showing a message: "No model card. Ask model author to add a README.md to this repo by tagging them on the Forum." and a button to "Contribute a Model Card". On the right, the "Hosted inference API" section shows 133,095 downloads last month with a line graph, the task "Token Classification", a text input "My name is Wolfgang and I live in Berlin", and a "Compute" button. Below the input, it states "This model is currently loaded and running on the Inference API." and provides a "JSON Output" link and a "Maximize" button.

Hugging Face

Search models, datasets, users...

Models Datasets Resources Solutions Pricing Log In Sign Up

dbmdz **bert-large-cased-finetuned-conll03-english** like 0

Token Classification PyTorch TensorFlow JAX Rust Transformers bert

Model card Files and versions Train Deploy Use in Transformers

No model card

Ask model author to add a README.md to this repo by tagging them on the Forum

Contribute a Model Card

Downloads last month  
133,095

Hosted inference API

Token Classification

My name is Wolfgang and I live in Berlin Compute

This model is currently loaded and running on the Inference API.

JSON Output Maximize

# Sequential Labelling is Hard

## ⚡ Hosted inference API ⓘ

🔍 Token Classification

My name is Robert Paulson and I live in Wilmington, Delaware

Compute

Computation time on cpu: 0.2724 s

My name is Robert Paulson **PER** and I live in Wilmington **LOC**, Delaware **LOC**

# Sequential Labelling is Hard

## ⚡ Hosted inference API ⓘ

🔍 Token Classification

Jesus of nazareth was born in Bethlehem

Compute

Computation time on cpu: 0.2496 s

Jesus **PER** of nazareth **LOC** was born in Bethlehem **LOC**

# Sequential Labelling is Hard

## ⚡ Hosted inference API ⓘ

🔍 Token Classification

Han Xiao thought Hugging Face was the competition of Jina

Compute

Computation time on cpu: 0.2628 s

Han Xiao **PER** thought Hugging Face **MISC** was the competition of Jina **PER**

# Sequential Labelling is Hard

## ⚡ Hosted inference API ⓘ

🔍 Token Classification

Han Xiao did not like Hugging Face much, that is why he built Jina

Compute

Computation time on cpu: 0.42 s

Han Xiao **PER** did not like Hugging **MISC** Face **PER** much, that is why he built

Jin **ORG** a **MISC**



# Sequential Labelling is Hard

## ⚡ Hosted inference API ⓘ

🔍 Token Classification

I love that Julia replaces standard classes by multiple dispatch

Compute

Computation time on cpu: 0.21359999999999998 s

I love that **Julia** **PER** replaces standard classes by multiple dispatch

# Sequential Labelling: Notation

- Let  $\Sigma := w_1, \dots, w_J$  be a set of words
- Let  $\Lambda := c_1, \dots, c_K$  be a set of labels
- A sentence is an element from  $\Sigma^*$  which is the Kleene closure of  $\Sigma$ 
  - The Kleene closure of a set  $\Sigma$  is defined as the set containing all possible strings of arbitrary length made up with elements in  $\Sigma$ ,  $\epsilon$  is used for the empty string
  - $\Sigma = a, b$  then  $\Sigma^* = \epsilon, a, b, aa, ab, ba, bb, aaa, aab, bab, bba, \dots$
  - $\Sigma = ab, c$  then  $\Sigma^* = \epsilon, ab, c, abab, abc, cab, cc, ababab, ababc, \dots$

# Sequential Labelling: Notation

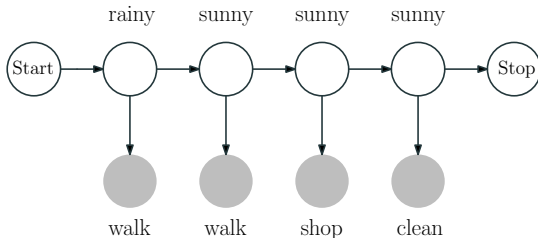
- A Hidden Markov Model (HMM) is a probabilistic model that defines a probability distribution over input/output pairs as follows:

$$P(X_1 = x_1, \dots, X_N = x_N, Y_1 = y_1, \dots, Y_N = y_N) = P_{\text{init}}(y_1 | \text{start}) \cdot \left( \prod_{i=1}^{N-1} P_{\text{trans}}(y_{i+1} | y_i) \right) \cdot P_{\text{final}}(\text{stop} | y_N) \cdot \prod_{i=1}^N P_{\text{emiss}}(x_i | y_i)$$

- The previous formula tells us how to compute the probability of a pair if we have the initial, transition, final and emission probabilities, but how do we find such probabilities?

# HMM Example

$$(x, y) = ([\text{walk}, \text{walk}, \text{shop}, \text{clean}], [\text{rainy}, \text{sunny}, \text{sunny}, \text{sunny}])$$



$$\begin{aligned} P([walk, walk, shop, clean], [rainy, sunny, sunny, sunny]) = \\ P_{init}(rainy|start) \cdot P_{trans}(sunny|rainy) \cdot P_{trans}(sunny|sunny) \cdot P_{trans}(sunny|sunny) \cdot P_{final}(stop|sunny) \cdot \\ P_{emiss}(walk|rainy) \cdot P_{emiss}(walk|sunny) \cdot P_{emiss}(shop|sunny) \cdot P_{emiss}(clean|sunny) \end{aligned}$$

# HMM Parameters in memory

Parameter	#Floats
$P_{\text{init}}(c \text{start})$	$ \Lambda $
$P_{\text{trans}}(c \hat{c})$	$ \Lambda ^2$
$P_{\text{final}}(\text{stop} c)$	$ \Lambda $
$P_{\text{emiss}}(w c)$	$ \Sigma  \cdot  \Lambda $

# HMM Properties

- **Independence of Previous States:** the probability of being in a given state at position  $i$  only depends on the state of the previous position  $i - 1$  (this defines a first order Markov chain)

$$P(Y_i = y_i | Y_{i-1} = y_{i-1}, Y_{i-2} = y_{i-2}, \dots, Y_1 = y_1) = P(Y_i = y_i | Y_{i-1} = y_{i-1})$$

- **Homogeneous Transition:** the probability of making a transition from state  $c_l$  to state  $c_k$  is independent of the particular sequence position

$$P(Y_i = c_k | Y_{i-1} = c_l) = P(Y_t = c_k | Y_{t-1} = c_l)$$

- **Observation Independence:** the probability of observing  $X_i = x_i$  at position  $i$  is fully determined by the state  $Y_i$  at that position. The probability is also independent of the particular position

$$P(X_i = x_i | Y_1 = y_1, \dots, Y_i = y_i, \dots, Y_N = y_N) = P(X_i = x_i | Y_i = y_i)$$

$$P(X_i = w_j | Y_i = c_k) = P(X_t = w_j | Y_t = c_k)$$

- Posterior decoding

$$y_i^* = \arg \max_{y_i \in \Lambda} P(Y_i = y_i | X_1 = x_1, \dots, X_N = x_N)$$

- Viterbi decoding

$$y^* = \arg \max_{y \in \Lambda^N} P(Y_1 = y_1, \dots, Y_N = y_N | X_1 = x_1, \dots, X_N = x_N)$$

# Learning the Parameters of an HMM

- HMM parameters can be learnt by counting the different *events* we encounter in the training data\*

Parameter	Value
$c_{\text{init}}(c_k)$	$\sum_{m=1}^M 1(y_1^m = c_k)$
$P_{\text{init}}(c_k   \text{start})$	$\frac{c_{\text{init}}(c_k)}{\sum_{l=1}^L c_{\text{init}}(c_l)}$
$c_{\text{trans}}(c_k, c_l)$	$\sum_{m=1}^M \sum_{i=1}^N 1(y_i^m = c_k \wedge y_{i-1}^m = c_l)$
$P_{\text{trans}}(c_k   c_l)$	$\frac{c_{\text{trans}}(c_k, c_l)}{\sum_{p=1}^K c_{\text{trans}}(c_p, c_l) + c_{\text{final}}(c_l)}$
$c_{\text{final}}(c_k)$	$\sum_{m=1}^M 1(y_N^m = c_k)$
$P_{\text{final}}(\text{stop}   c_l)$	$\frac{c_{\text{final}}(c_l)}{\sum_{k=1}^K c_{\text{trans}}(c_k, c_l) + c_{\text{final}}(c_l)}$
$c_{\text{emiss}}(w_j, c_k)$	$\sum_{m=1}^M \sum_{i=1}^N 1(x_i^m = w_j \wedge y_i^m = c_k)$
$P_{\text{emiss}}(c_k   c_l)$	$\frac{c_{\text{emiss}}(w_j, c_k)}{\sum_{q=1}^J c_{\text{emiss}}(w_q, c_k)}$

\*  $M$ : number of samples



# Forward Backward Algorithm

- Forward quantity

$$f(i, x, c) := P(X_1 = x_1, \dots, X_i = x_i, Y_i = c)$$

- Backward quantity

$$b(i, x, c) := P(X_{i+1} = x_{i+1}, \dots, X_N = x_N | Y_i = c)$$

# Efficient Forward Quantity Computations

- If we define  $f(i, x, c) := P(X_1 = x_1, \dots, X_i = x_i, Y_i = c)$  then, the forward quantity at position  $i$  can be written as:

$$f(i, x, c) = P_{\text{emiss}}(x_i | c) \cdot \left( \sum_{\tilde{c} \in \Lambda} P_{\text{trans}}(c | \tilde{c}) \cdot f(i-1, x, \tilde{c}) \right)$$

# Recurrence Rule for the Forward Quantities

$$f(i, x, c) := P(X_1 = x_1, \dots, X_i = x_i, Y_i = c)$$

$$\begin{aligned} P(y_i, x_{1:i}) &= \sum_{y_{i-1} \in \Lambda} P(y_i, y_{i-1}, x_{1:i}) = \\ &\sum_{y_{i-1} \in \Lambda} P(y_i, y_{i-1}, x_{1:i-1}, x_i) = \\ &\sum_{y_{i-1} \in \Lambda} P(x_i | y_i, y_{i-1}, x_{1:i-1}) \cdot P(y_i, y_{i-1}, x_{1:i-1}) = \\ &\sum_{y_{i-1} \in \Lambda} P(x_i | y_i, y_{i-1}, x_{1:i-1}) \cdot P(y_i | y_{i-1}, x_{1:i-1}) \cdot P(y_{i-1}, x_{1:i-1}) = \\ &\sum_{y_{i-1} \in \Lambda} P(x_i | y_i) \cdot P(y_i | y_{i-1}) \cdot P(y_{i-1}, x_{1:i-1}) = \\ &P(x_i | y_i) \cdot \sum_{y_{i-1} \in \Lambda} P(y_i | y_{i-1}) \cdot P(y_{i-1}, x_{1:i-1}) \end{aligned}$$

# Forward Quantities Definition

- Let us consider a sequence  $x$  with  $N$  words
- For every state  $c$  let us define:

$$f(1, x, c) = P_{\text{init}}(c_k | \text{start}) \cdot P_{\text{emiss}}(x_1 | c_k)$$

- For every position  $i = 2$  up to  $N$

$$f(i, x, c) = P_{\text{emiss}}(x_i | c) \cdot \left( \sum_{\tilde{c} \in \Lambda} P_{\text{trans}}(c | \tilde{c}) \cdot f(i-1, x, \tilde{c}) \right)$$

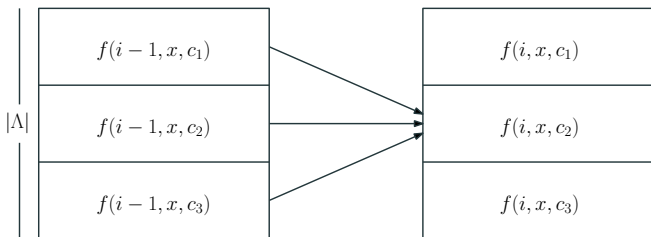
- For position  $N+1$

$$f(N+1, \text{stop}) = \sum_{c_l \in \Lambda} P_{\text{final}}(\text{stop} | c) \cdot f(N, x, c)$$

$$f(i, x, c) = P_{\text{emiss}}(x_i | c) \cdot \left( \sum_{\tilde{c} \in \Lambda} P_{\text{trans}}(c | \tilde{c}) \cdot f(i-1, x, \tilde{c}) \right)$$

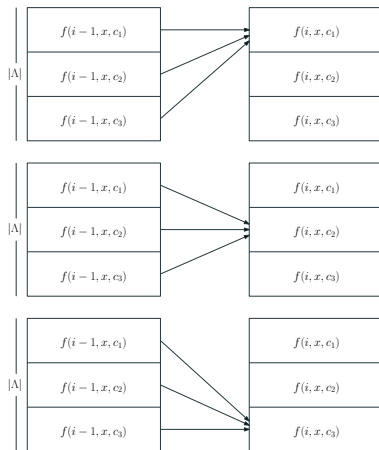
$\Lambda$						
	$f(1, x, \cdot)$	$f(2, x, \cdot)$		$f(i-1, x, \cdot)$	$f(i, x, \cdot)$	$f(N, x, \cdot)$

$$f(i, x, c) = P_{\text{emiss}}(x_i | c) \cdot \left( \sum_{\tilde{c} \in \Lambda} P_{\text{trans}}(c | \tilde{c}) \cdot f(i-1, x, \tilde{c}) \right)$$



# Forward Quantity Given the Previous Forward Quantity

$$f(i, x, c) = P_{\text{emiss}}(x_i | c) \cdot \left( \sum_{\tilde{c} \in \Lambda} P_{\text{trans}}(c | \tilde{c}) \cdot f(i-1, x, \tilde{c}) \right)$$



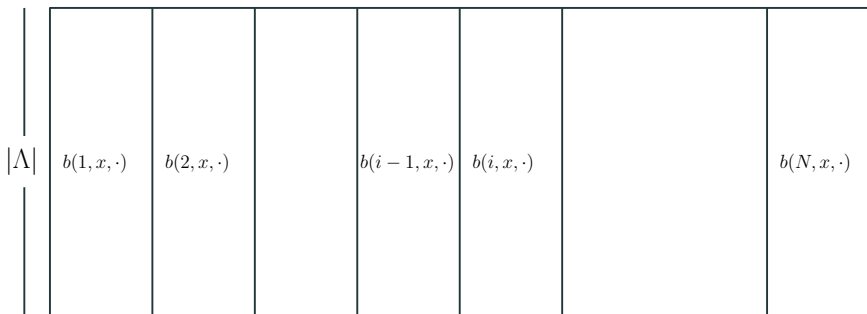
$$b(i, x, c) := P(X_{i+1} = x_{i+1}, \dots, X_N = x_N | Y_i = c)$$

Exercise!



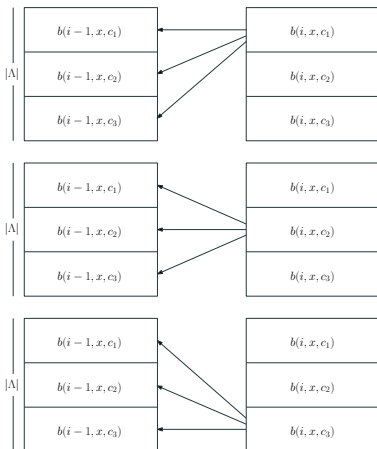
# Backward Quantities

$$b(i, x, c) := \sum_{\tilde{c} \in \Lambda} P_{\text{trans}}(\tilde{c} | c) \cdot P_{\text{emiss}}(x_{i+1} | \tilde{c}) \cdot b(i + 1, x, \tilde{c})$$



# Backward Quantities

$$b(i, x, c) := \sum_{\tilde{c} \in \Lambda} P_{\text{trans}}(\tilde{c}|c) \cdot P_{\text{emiss}}(x_{i+1}|\tilde{c}) \cdot b(i+1, x, \tilde{c})$$



# Forward Backward Algorithm

- Proposition I:  $P(X = x, Y_i = c) = f(i, x, c) \cdot b(i, x, c)$
- Proposition II:  $P(X = x) = \sum_{c \in \Lambda} f(i, x, c) \cdot b(i, x, c)$
- Proposition III:  $P(Y_i = c | X = x) = \frac{f(i, x, c) \cdot b(i, x, c)}{P(X=x)}$
- Proposition IV (Posterior decoding):

$$y^* = \arg \max_{c \in \Lambda} P(Y_i = c | X_1 = x_1, \dots, X_N = x_N) = \arg \max_{c \in \Lambda} f(i, x, c) \cdot b(i, x, c)$$

# Forward Backward Algorithm

```
forward_backward( $x, \Lambda, P_{\text{init}}, P_{\text{trans}}, P_{\text{final}}, P_{\text{emiss}}$ )  
  # Forward pass  
  for  $c_k \in \Lambda$ :  
    forward( $1, x, c_k$ ) =  $P_{\text{init}}(c_k | \text{start}) \cdot P_{\text{emiss}}(x_1 | c_k)$   
  for  $i = 1$  to  $N$ :  
    for  $\tilde{c} \in \Lambda$ :  
      forward( $i, x, \tilde{c}$ ) =  $\left( \sum_{c_k \in \Lambda} P_{\text{trans}}(\tilde{c} | c_k) \cdot \text{forward}(i - 1, x, c_k) \right) P_{\text{emiss}}(x_i | \tilde{c})$   
  
  # Backward pass  
  for  $c_k \in \Lambda$ :  
    backward( $N, c_k$ ) =  $P_{\text{final}}(\text{stop} | c_k)$   
  for  $i = N - 1$  to  $1$ :  
    for  $\tilde{c} \in \Lambda$ :  
      backward( $i, x, \tilde{c}$ ) =  $\sum_{c_k \in \Lambda} P_{\text{trans}}(c_k | \tilde{c}) \cdot P_{\text{emiss}}(x_{i+1} | c_k) \cdot \text{backward}(i + 1, x, c_k)$ 
```