Sequence Classification using SENNA

Nils Reimers

http://www.deeplearning4nlp.com

Overview

TECHNISCHE UNIVERSITÄT DARMSTADT

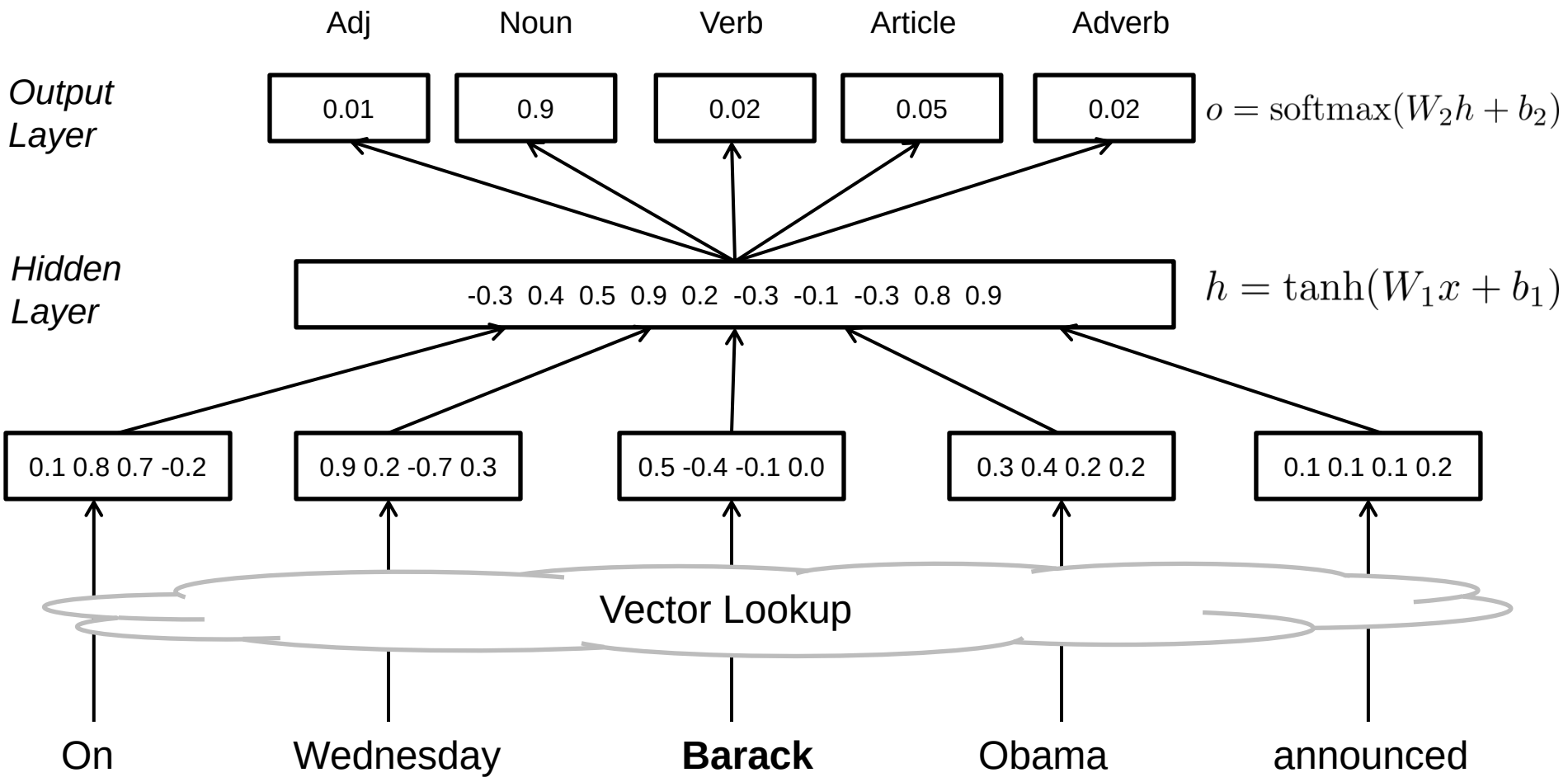
- This session introduces an architecture that can be used for word and sequence classification, e.g. POS-tagging, NER, Chunking.

- The architecture is known as SENNA and was proposed by:
  - Collobert et al., 2011, Natural Language Processing (Almost) from Scratch

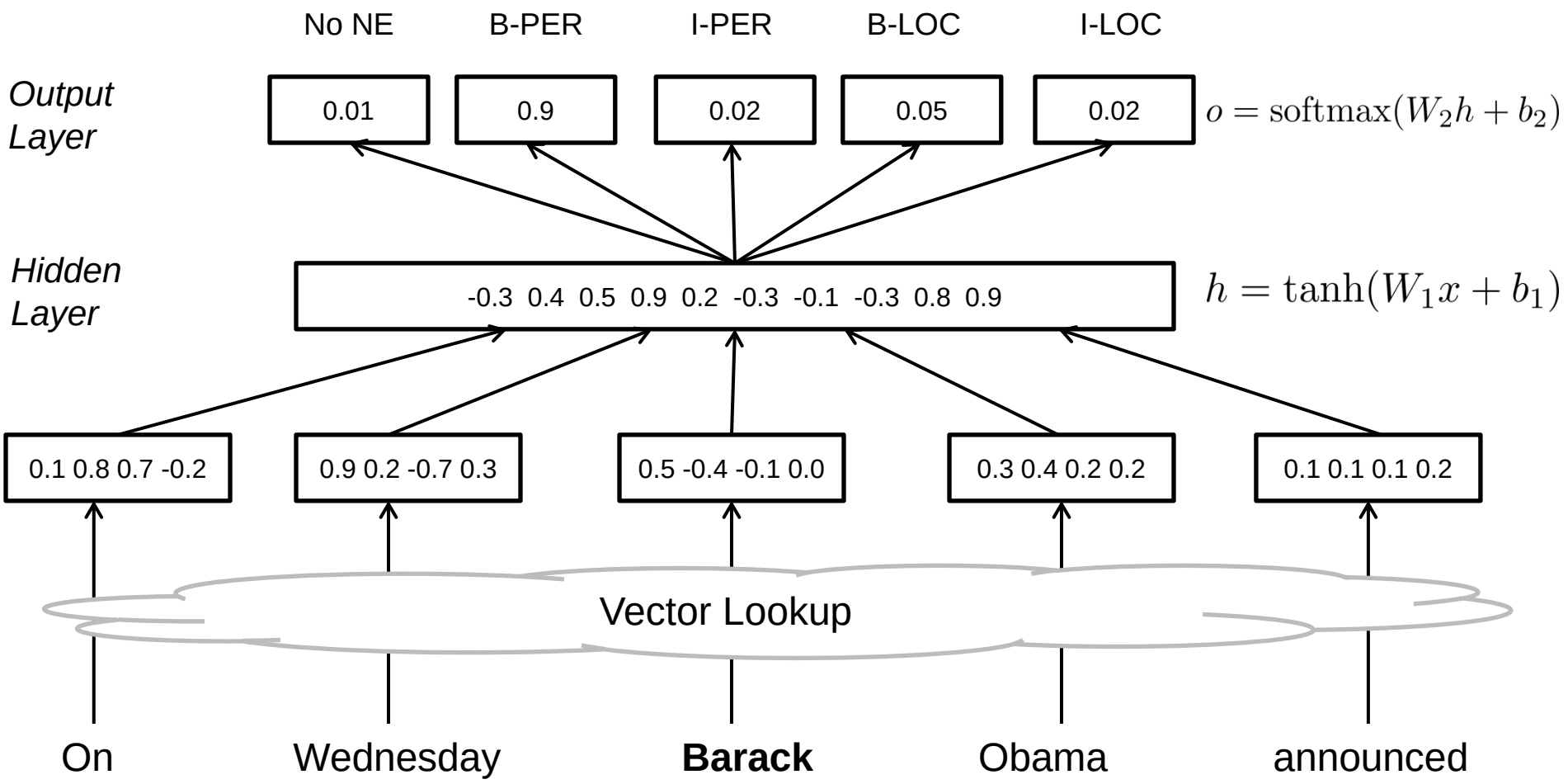
- The paper proposes two different architectures:
  - Window-Approach: Suitable if the necessary information is in the direct context of the target word (e.g. NER, POS)
  - Sentence-Approach: Takes the complete sentence into account (e.g. SRL)

- The paper proposes two different learning objectives:
  - Isolated-Tag-Criterion: The output label is optimized for each token individually
  - Sentence-Tag-Criterion: A Hidden Markov Model is added on top

POS-Tagging (Window-Approach + Isolated Tag Criterion)

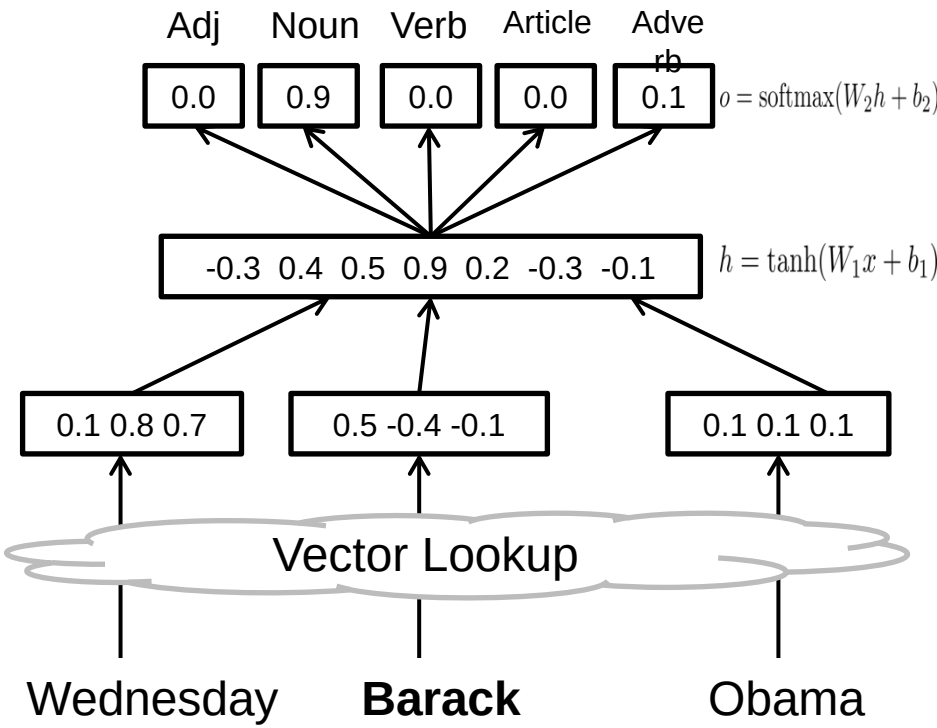


Can be easily extended to NER (using BIO encoding)



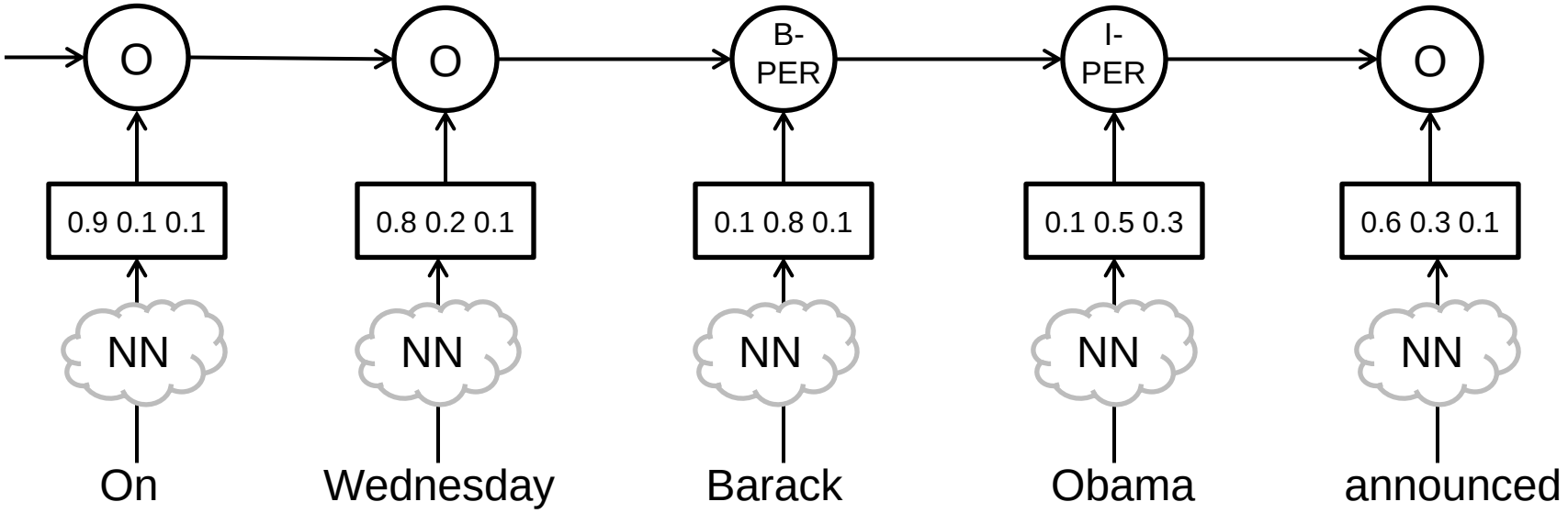
SENNA Step-by-Step

1. Select a target word, e.g. **Barack**
2. Create a window of *n* tokens around the target. A window-size=1 would create the tuple (Wednesday, **Barack**, Obama)
   - Use special PADDING token for tokens at sentence border, e.g. (PADDING, **Wednesday**, Barack)
3. Map each token to its word embedding (e.g. 100 dim. word embedding)
4. Concatenate the 3x100 dim. vectors and feed the 300 dim. vector into a dense hidden layer and apply tanh-activation function
5. Take the output of the hidden layer, feed it into a dense layer and apply the softmax-activiation function



Excursion: Sentence-Tag-Criterion

- Collobert et al. also defines a Sentence-Tag-Criterion (STC), which adds a Hidden Markov Model on top of the neural network
- The tagging sequence of the complete sentence is optimized
- Superior results, but currently no out-of-the-box support in Keras



Implementing this in Python + Keras

TECHNISCHE UNIVERSITÄT DARMSTADT

- Code consists of two parts:
  - preprocess.py: Creates matrices that we can feed into the network
  - POS.py / NER.py: Our Keras Neural Network implementation

- Bringing the data into the right format is often harder than building and training the network :/

Preprocess.py

- We will create a matrix for the tokens, for the casing of the words, and for the labels
- Token Matrix:

$$\begin{bmatrix} PADDING & On & Wednesday \\ On & Wednesday & Barack \\ Wednesday & Barack & Obama \\ & \dots & \\ PADDING & Second & sentence \\ Second & sentence & . \end{bmatrix}$$

- Map each token to its index in the word embeddings matrix
- Maybe convert words to lower case

$$\begin{bmatrix} 0 & 15 & 18 \\ 15 & 18 & 27 \\ 18 & 27 & 3 \\ & \dots & \\ 0 & 54 & 72 \\ 54 & 72 & 4 \end{bmatrix}$$

Preprocess.py

TECHNISCHE UNIVERSITÄT DARMSTADT

- We apply the same technique for word casing information
- Case Matrix:

$$\begin{bmatrix} PADDING & initialUpper & initialUpper \\ initialUpper & initialUpper & initialUpper \\ initialUpper & initialUpper & initialUpper \\ & \dots & \\ PADDING & initialUpper & allLower \\ initialUpper & allLower & other \end{bmatrix}$$

- Map each casing information to the index in the embedding lookup
- The casing embedding matrix is a hot-one encoding matrix

$$\begin{bmatrix} 0 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \\ & \dots & \\ 0 & 2 & 1 \\ 2 & 1 & 5 \end{bmatrix}$$

Preprocess.py

TECHNISCHE UNIVERSITÄT DARMSTADT

- The labels are converted to an integer vector, each entry specifying the label for the specific window
- Labels:

$$\begin{bmatrix} O \\ B-PER \\ I-PER \\ \dots \\ O \\ O \end{bmatrix}$$

$$\downarrow$$

- Each label is mapped to an integer

$$\begin{bmatrix} 0 \\ 1 \\ 2 \\ \dots \\ 0 \\ 0 \end{bmatrix}$$

Preprocess.py – Final Result