Auto-correct and Dynamic Programming

Auto-correct

is an application that changes misspelled words into the correct ones.

- Example: Happy birthday deah friend! ==> dear

- How it works:

    1.Identify a misspelled word

    2. Find strings n edit distance away

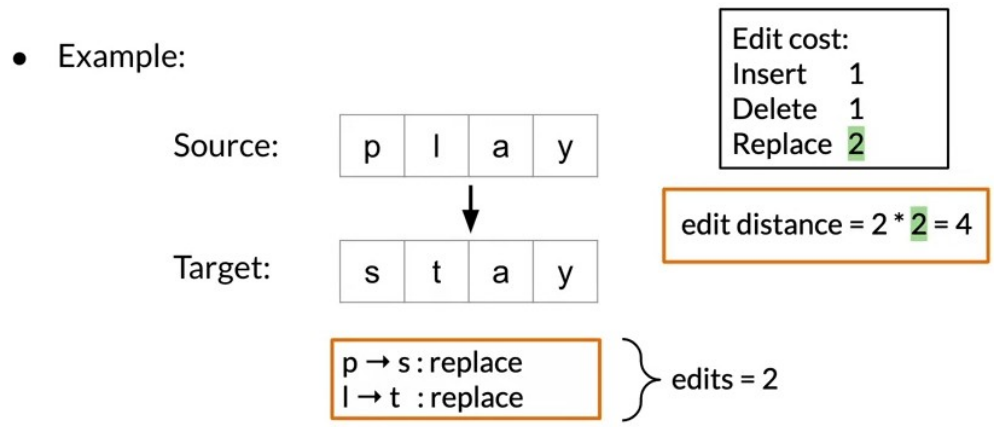
    3. Filter candidates

    4. Calculate word probabilities

Minimum edit distance

- Evaluate the similarity between 2 strings.
- Minimum number of edits needed to transform 1 string into another.

- the algorithm try to minimize the edit cost.



Applications:

- Spelling correction
- document similarity
- machine translation
- DNA sequencing

Part of Speech Tagging and Hidden Markov Models

Part of Speech Tagging

The category of words or the lexical terms in the language.

Tags: Noun, Verb, adjective, preposition, adverb,...

Part of speech tags:

lexical term	tag	example
noun	NN	something, nothing
verb	VB	learn, study
determiner	DT	the, a
w-adverb	WRB	why, where
...	...	

Why not learn something ?

WRB RB VB NN .

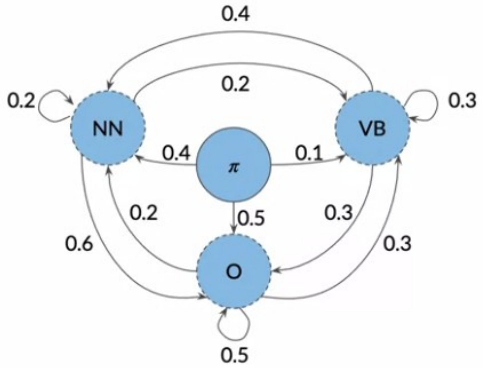
Applications:

- Named entities
- Co-reference resolution
- Speech recognition

Markov Chains

A stochastic model describing a sequence of possible events.

Transition probabilities

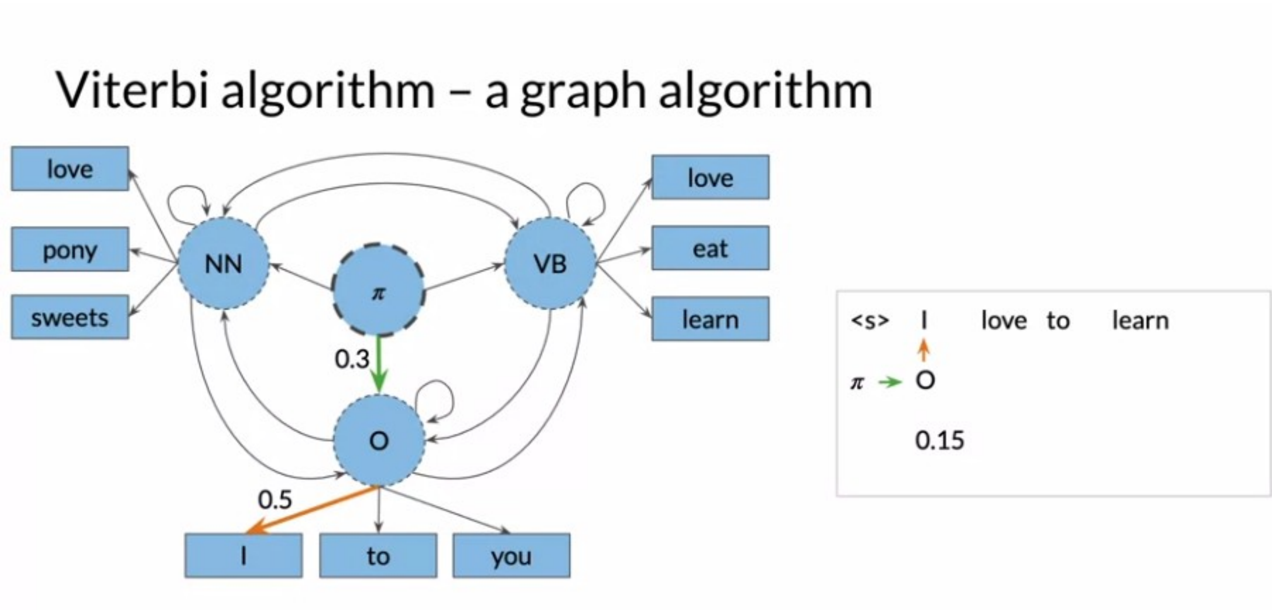


$A =$

	NN	VB	O
π (initial)	0.4	0.1	0.5
NN (noun)	0.2	0.2	0.6
VB (verb)	0.4	0.3	0.3
O (other)	0.2	0.3	0.5

The Viterbi Algorithm

A graph algorithm that finds the sequence of hidden states or parts of speech tags that have the highest probability for a sequence.



Given your transition and emission probabilities, we first populates and then use the auxiliary matrices C and D

matrix C holds the intermediate optimal probabilities.

$$c_{i,j} = \max_k c_{k,j-1} * a_{k,i} * b_{i,cindex(w_j)}$$

matrix D holds the indices of the visited states (tags).

$$d_{i,j} = \operatorname{argmax}_k c_{k,j-1} * a_{k,i} * b_{i,cindex(w_j)}$$

$C =$

	w_1	w_2	w_3	w_4	w_5
t_1	0.25	0.125	0.025	0.0125	0.01
t_2	0.1	0.025	0.05	0.01	0.003
t_3	0.3	0.05	0.025	0.02	0.0000
t_4	0.2	0.1	0.000	0.0025	0.0003

$$\boxed{s = \operatorname{argmax}_i c_{i,K} = 1}$$

C[1,5] = 0.01

D[1,5] = 1

Word Embeddings

Basic Word Representations

1. **Integers**

    + Simple

    - Ordering: no semantic sense

2. **One-hot vectors**

    + Simple

    + No implied ordering

    - Takes a lot of time and space

    - No embedded meaning

3. **Word embedding vectors**

    + Low dimension

    - e.g. semantic distance: forest ≈ tree

    + Embed meaning

    - e.g. analogies: Paris:France :: Rome:?

Basic word embedding methods

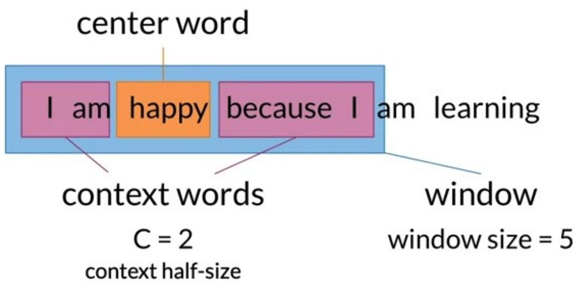
- **word2vec** (Google, 2013)
    - Continuous bag-of-words (CBOW): Which predict the missing word just giving the surround word.
    - Continuous skip-gram (SGNS): which does the reverse of the CBOW method, SGNS learns to predict the word surrounding a given input word.
- **Global Vectors (GloVe)** (Stanford, 2014)
- **FastText** (Facebook, 2016): based on the skip-gram model.

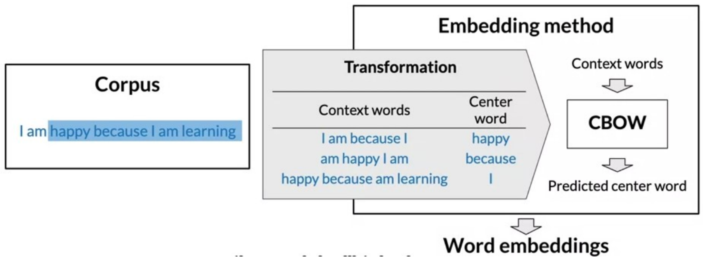
Advanced word embedding methods

- **BERT** (Google, 2018)
- **ELMo** (Allen Institute for AI, 2018)
- **GPT-2** (OpenAI, 2018)

  Note: Tunable pre-trained models.

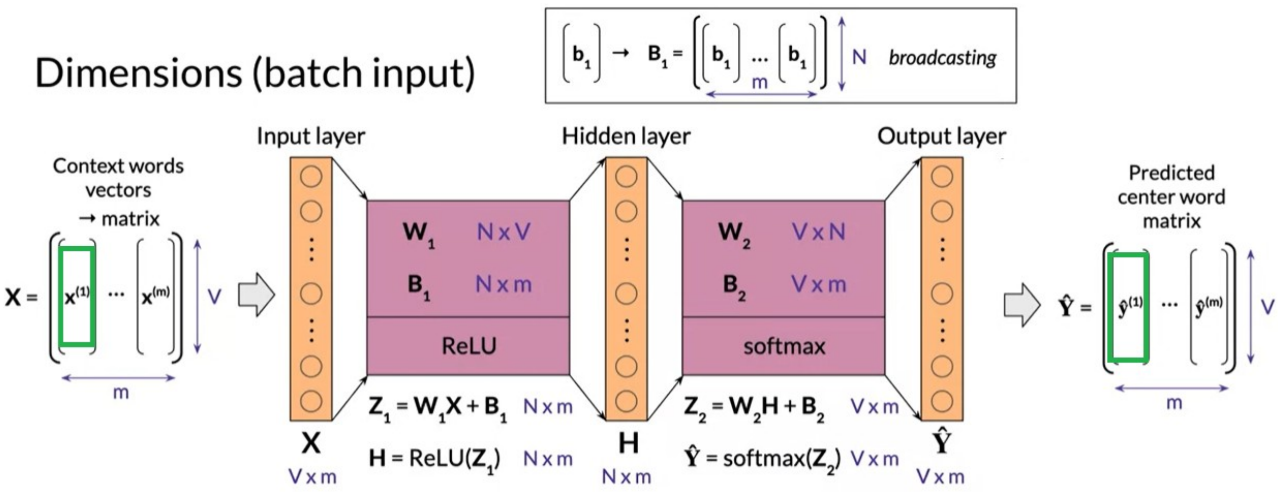
Continuous Bag-of-Words Model



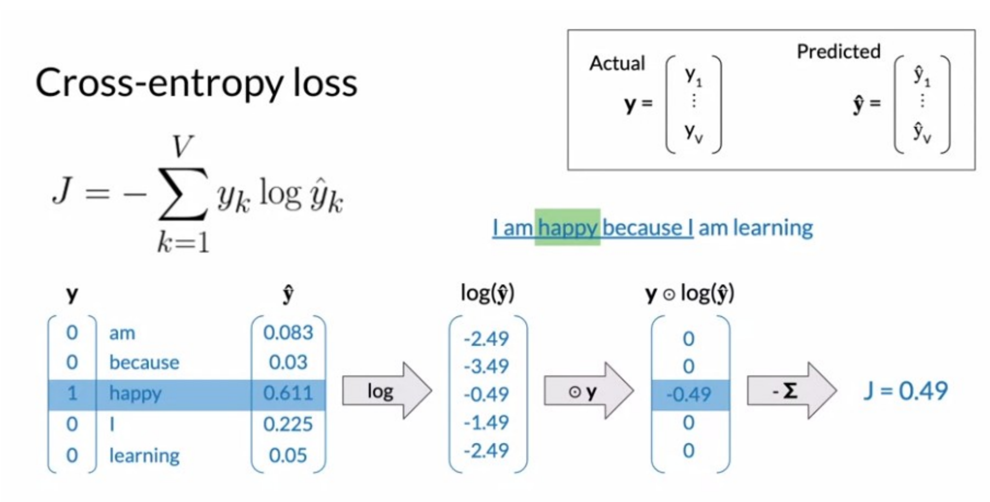


Architecture

- CBOW model is based on the shallow dense neural network with an input layer, a single hidden layer, and output layer.



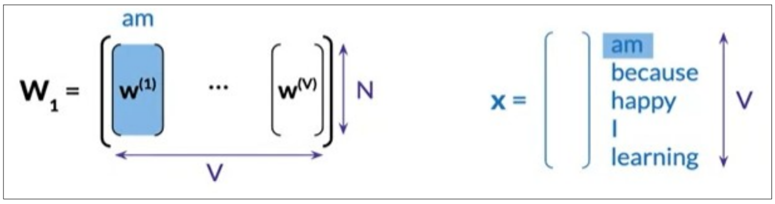
Cost Function: Cross-entropy loss (log loss)



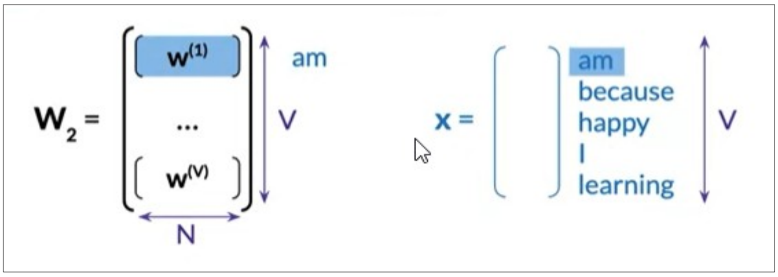
Extracting Word Embedding Vectors

After we have trained the neural network, we can extract three alternative word embedding representations.

i. Consider each column of W_1 as the column vector embedding vector of a word of the vocabulary



ii. Use each row of W_2 as the word embedding row vector for the corresponding word.



iii. Average W_1 and the transpose of W_2 to obtain W_3, a new n by v matrix.