Lecture 6: Neural Networks for Representing Word Meaning

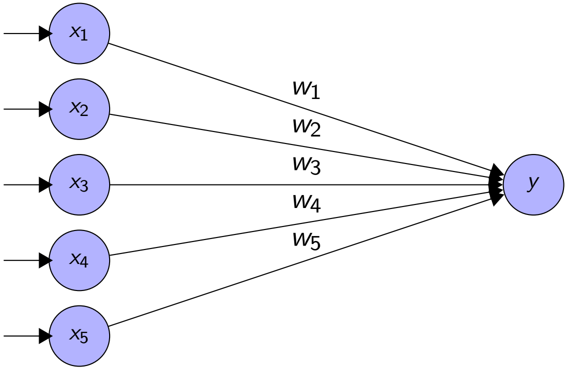
Mirella Lapata

School of Informatics
University of Edinburgh
`mlap@inf.ed.ac.uk`

February 7, 2017

Logistic Regression

- **Input** is a feature vector, **output** is one (binary classification) or many (multinomial distribution)
- Weights matrix (or vector) directly connects inputs and output
- Trained by gradient descent (neural network; no hidden units).



Logistic Functions

Logistic Function

$$f(x) = \frac{L}{1+e^{-k(x-x_0)}}$$

Sigmoid

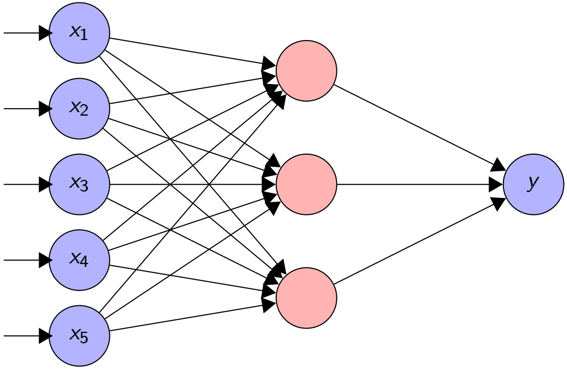
$$f(x) = \frac{1}{1+e^{-x}}$$

Softmax function

$$P(y=j|\mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w_j}}}{\sum_{k=1}^{K} e^{\mathbf{x}^\top \mathbf{w_k}}}$$

- e natural logarithm base, x_0 the x-value of the sigmoid's midpoint, L the curve's maximum value, k steepness of curve
- Sigmoid is special case of the logistic function
- Softmax is a generalization of the logistic function
- $\mathbf{x}^\mathsf{T}\mathbf{w}$ denotes the inner product of $\mathbf{x}$ and $\mathbf{w}$

Neural Networks



W: input weights, matrix
I: input signal, feature vector (one per example)
Activation function: sigmoid, tanh

Training of Neural Networks

Forward Pass

- Input signal is presented first
- Hidden layer state is computed (vector times matrix operation and non-linear activation)
- Outputs are computed (vectors times matrix operation and usually non-linear activation)

Backpropagation

- To train the network, we need to compute gradient of the error
- The gradients are sent back using the same weights that were used in the forward pass

Gradient Descent

- Batch gradient descent: compute gradient from batch of N training examples
- Stochastic gradient descent: compute the gradient from 1 training example each time
- Minibatch: compute the gradient from minibatch of M training examples ($M > 1$, $M < N$)

Training of Neural Networks

Learning Rate:

- Controls how much we change the weights
- Too little value will result in long training time, too high value will erase previously learned patterns
- We start with high learning rate and reduce it during training

Training Epochs: several passes over the training data are often performed (epoch: number of iterations over the data set in order to train the neural network).

Regularization:

- Network often overfits (it fails to generalize at test time)
- High weights are used to model only some small subset of data
- We can try to force the weights to stay small during training to avoid this problem (L1 & L2 regularization)

Model Parameters

Choice of the **hyper-parameters** has to be done manually:

- Type of activation function
- Choice of architecture (how many hidden layers, their sizes)
- Learning rate, number of training epochs
- What features are presented at the input layer
- How to regularize

N-grams

N-gram-based language models are often used to compute the probability of a sentence W:

$$P(W) = \prod_i (w_i | w_1 \ldots w_{i-1})$$

Often simplified to trigrams:

$$P(W) = \prod_i (w_i | w_{i-2}, w_{i-1})$$

$$\begin{aligned} P(this\ \ is\ \ a\ \ sentence) &= P(this) \times P(is|this) \times P(a|this, is) \times \\ &\ P(sentence|is, a) \\ \\ P(a|this,\ \ is) &= \frac{C(this\ \ is\ \ a)}{C(this\ \ is)} \end{aligned}$$

One-hot Representations

Simple way how to encode discrete concepts, such as words; also known as 1-of-N where N would be the size of the vocabulary.

vocabulary = (Monday, Tuesday, is, a, today)

$$\begin{aligned} \text{Monday} &= [1\ 0\ 0\ 0\ 0] \\ \text{Tuesday} &= [0\ 1\ 0\ 0\ 0] \\ \text{is} &= [0\ 0\ 1\ 0\ 0] \\ \text{a} &= [0\ 0\ 0\ 1\ 0] \\ \text{today} &= [0\ 0\ 0\ 0\ 1] \end{aligned}$$

Bag-of-Words Representations

Ignores word order, sum of one-hot vectors. Can be extended to bag-of-N-grams to capture local ordering of words.

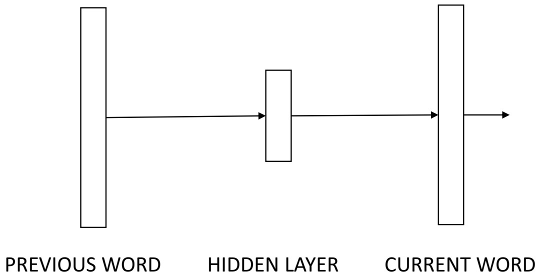
vocabulary = (Monday, Tuesday, is, a, today)

$$\begin{aligned}
\text{Monday} &= [1\ 0\ 0\ 0\ 0] \\
\text{Tuesday} &= [0\ 1\ 0\ 0\ 0] \\
\text{is} &= [0\ 0\ 1\ 0\ 0] \\
\text{a} &= [0\ 0\ 0\ 1\ 0] \\
\text{today} &= [0\ 0\ 0\ 0\ 1]
\end{aligned}$$

vocabulary = (Monday, Tuesday, is, a, today)

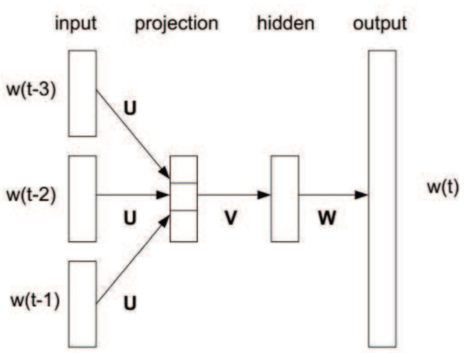
$$\begin{aligned}
\text{Monday Monday} &= [2\ 0\ 0\ 0\ 0] \\
\text{today is a Monday} &= [1\ 0\ 1\ 1\ 1] \\
\text{today is a Tuesday} &= [0\ 1\ 1\ 1\ 1] \\
\text{is a Monday today} &= [1\ 0\ 1\ 1\ 1]
\end{aligned}$$

A Basic Neural Language Model



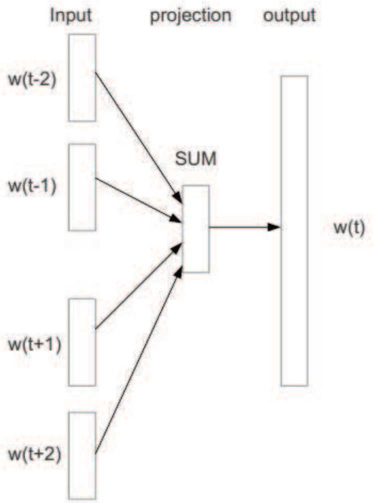
- Bigram neural language model
- Previous word predicts current word via hidden layer
- As many outputs as there are words in the vocabulary
- Model learns compressed, continuous representations of words (usually the matrix of weights between input and hidden layer)

Another Neural Language Model



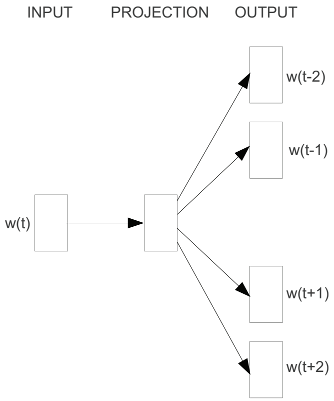
- Bengio et al. (2003, JMLR)
- Input layer, projection layer, hidden layer and output layer
- The projection layer is linear
- The hidden layer is non-linear
- Softmax at the output computes probability distribution over the whole vocabulary ($g(o) = \frac{e^{o_i}}{\sum_k e^{o_k}}$)
- Model is computationally very expensive

Continuous bag-of-words model (CBOW)



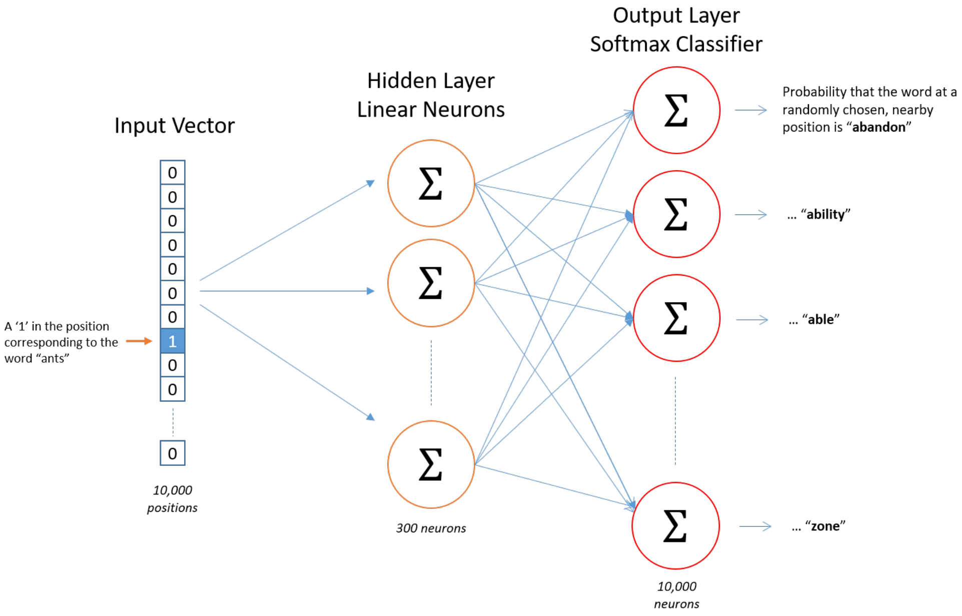
- Mikolov et al. (2013, ICLR)
- CBOW adds inputs from words within short window to predict the current word
- The weights for different positions are shared
- Computationally much more efficient than normal NNLM
- The hidden layer is just linear

Skip-gram NNLM

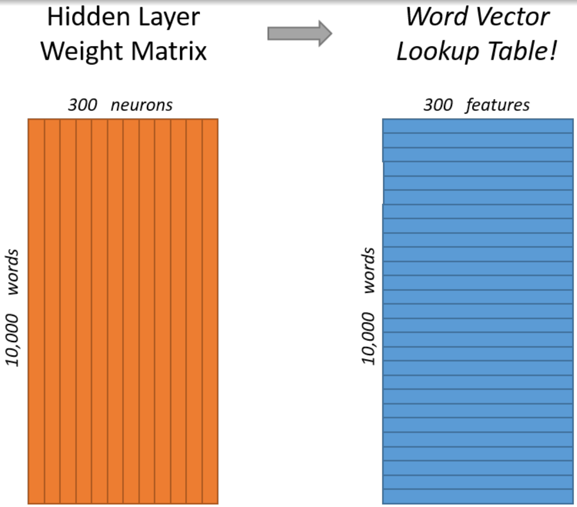


- We can reformulate the CBOW model by predicting surrounding words using the current word
- Find word representations useful for predicting surrounding words in a sentence or document.

Skip-gram NNLM

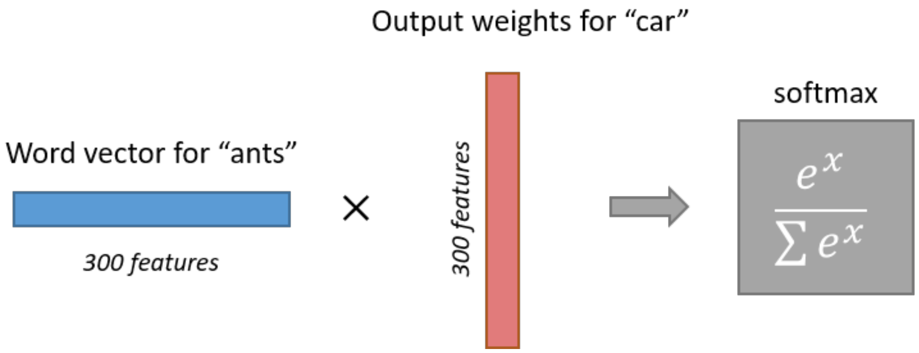


The Hidden Layer



We wish to learn word vectors with 300 features. Hidden layer is a weight matrix with 10,000 rows (one for every word in vocabulary) and 300 columns (one for every hidden neuron).

Output Weights



Skip-gram Details

- Each word $w \in W$ is associated with vector $v_w \in R^d$
- Each context $c \in C$ is associated with vector $v_c \in R^d$
- W words vocabulary, C contexts vocabulary
- d embedding dimensionality
- Vector components are latent (parameters to be learnt).
- Objective function maximizes the probability of corpus D (set of all word and context pairs extracted from text):

$$\underset{\theta}{\operatorname{argmax}} \prod_{(w,c) \in D} p(c|w;\theta)$$

Skip-gram Details

The basic Skip-gram formulation defines $p(c|w;\theta)$ using the softmax function:

$$p(c|w;\theta) = \frac{e^{v_c \cdot v_w}}{\sum_{c' \in C} e^{v_{c'} \cdot v_w}}$$

where v_c and v_w are vector representations for c and w and C is the set of all available contexts.

- We seek parameter values (i.e., vector representations for both words and contexts) such that the dot product $v_w \cdot v_c$ associated with "good" word-context pairs is maximized.
- The formulation is impractical due to the summation term $\sum_{c' \in C} e^{v_{c'} \cdot v_w}$ over all contexts c'.

Training

- Stochastic gradient descent and backpropagation
- It is useful to sub-sample the frequent words (e.g., *the, is, a*)
- Words are thrown out proportional to their frequency (makes things faster, reduces importance of frequent words like IDF)
- Non-linearity does not seem to improve performance of these models, thus the hidden layer does not use activation function
- **Problem:** very large output layer -size equal to vocabulary size, can easily be in order of millions (too many outputs to evaluate)
- **Solution:** negative sampling (also Hierarchical softmax)

Negative Sampling

- Instead of propagating signal from the hidden layer to the whole output layer, only the output neuron that represents the positive class + few randomly sampled neurons are evaluated
- The output neurons are treated as independent logistic regression classifiers
- This makes the training speed independent of the vocabulary size (can be easily parallelized)

Negative Sampling Objective

- Let D denote dataset of observed (w, c) pairs
- $p(D = 1|w, c)$ denotes probability that (w, c) came from D
- $p(D = 0|w, c) = 1 - p(D = 1|w, c)$ that it didn't

$$\underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} p(D = 1|w, c; \theta) \prod_{(w,c)\in D'} p(D = 0|w, c; \theta)$$

$$\underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log p(D = 1|w, c; \theta) + \sum_{(w,c)\in D'} \log(1 - p(D = 1|w, c; \theta))$$

$$\underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+e^{-v_c \cdot v_w}} + \sum_{(w,c)\in D'} \log \frac{1}{1+e^{v_c \cdot v_w}}$$

See Goldberg and Levy (2014) for full derivation.

Model Evaluation

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

- 20K questions, 6B words, 1M words vocabulary
- Find word closest to question word (e.g., *Greece*), consider it correct if it matches the answer.

Model Evaluation

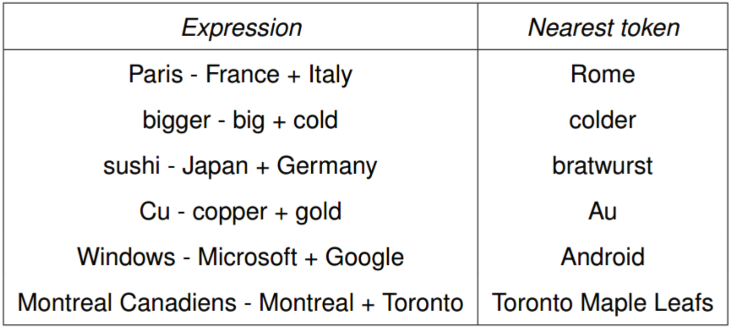
Models trained on the same data (640-dimensional word vectors).

Model Architecture	Semantic-Syntactic Word Relationship test set	
	Semantic Accuracy [%]	Syntactic Accuracy [%]
RNNLM	9	36
NNLM	23	53
CBOW	24	64
Skip-gram	55	59

Can you think of additional baseline comparisons?

Model Evaluation

Subtracting two word vectors, and add the result to another word.

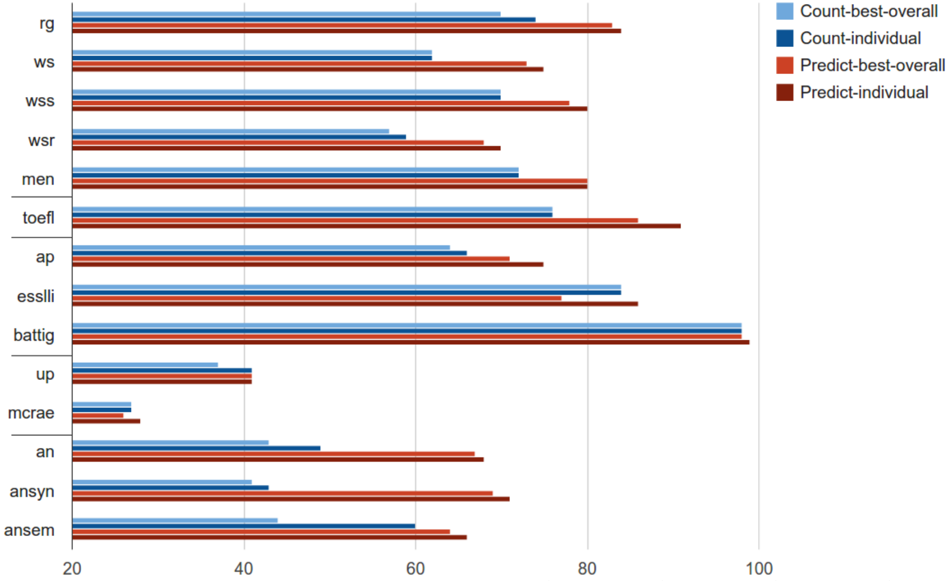


Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Check more examples out at:
`http://rare-technologies.com/word2vec-tutorial/#app`

Summary

Don't count predict! Baroni et al. (ACL, 2014)



Why Does Skip-gram Work?

- We don't really know!
- The objective tries to increase the quantity $v_w \cdot v_c$ for good word-context pairs, and decrease it for bad ones.
- Words that share many contexts will be similar to each other.
- Lots of parameters here too: contexts, subsampling, rare word pruning, dimension of vectors.

Resources

- `Word2vec` available at `https://code.google.com/p/word2vec/`
- Tool for training the word vectors using CBOW and skip-gram architectures, supports both negative sampling and hierarchical softmax
- Optimized for very large datasets ($>$billions of training words)
- Pre-trained on large datasets (100B words)