CS689: Computational Linguistics for Indian Languages
Term Embedding

Arnab Bhattacharya
arnabb@cse.iitk.ac.in

Computer Science and Engineering,
Indian Institute of Technology, Kanpur
http://web.cse.iitk.ac.in/~cs689/

2nd semester, 2023-24

Tue 10:30–11:45, Thu 12:00–13:15 at RM101/KD102

Embedding Models

- Models for *embedding* terms to vector spaces
- Uses of term vectors
  - Words can be compared, subtracted, found similarity of
  - Word similarity task: Are two words related?
  - Word analogy task: Given two related words and a third word, find the appropriate fourth word
- *Term-term context* matrix
  - Context window
  - Whether term i occurs within k number of terms upstream and downstream of term j

Term Co-occurrence and Context

- Global Vectors (GloVe)
- Global co-occurrence matrix X
- X_{ij} encodes how many times term i has appeared in the context of term j
- Sum of a row $X_i = \sum_{\forall j} X_{ij}$ denotes the *total* number of occurrences of term i
- *Probability* that term j occurs in the context of term i is

$$P_{ij} = P(j|i) = \frac{X_{ij}}{X_i}$$

Raw Probabilities and Ratio

- Raw counts X_{ij} or even raw probabilities P_{ij} may not be useful
- Depends a lot on the actual terms i and j
- Also, asymmetric
- *Ratio* is a better indicator of relevance

	$P(k\|ice)$	$P(k\|steam)$	$\frac{P(k\|ice)}{P(k\|steam)}$
k = solid	1.9e-4	2.2e-5	8.636
k = gas	6.6e-5	7.8e-4	0.084
k = water	3.0e-3	2.2e-3	1.363
k = fashion	1.7e-5	1.8e-5	0.944

- "solid" is more relevant to "ice" than "steam"
- "gas" is more relevant to "steam" than "ice"
- "water" is relevant to both "ice" and "steam"
- "fashion" is irrelevant to both "ice" and "steam"

Ratio of Probabilities

- Therefore, whether a term k is more relevant to term i than term j depends on the ratio P_{ik}/P_{jk}
- Hence, for *context vector* $\tilde{w}_k$ and *term vectors* w_i, w_j

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

- Vectors are of some dimensionality d
- Since ratio of probabilities is scalar

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

- Replacing probabilities by the same functional form

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}$$

Symmetry

- Functional form solution is $F = exp$
- Since $F(w_i^T \tilde{w}_k) = P_{ik}$

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$
$$w_i^T \tilde{w}_k + b_i = \log(X_{ik})$$

- Bias b_i encapsulates count of term i
- Symmetric: both terms i and k should be in the context of each other

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

- To avoid zero count problems

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(1 + X_{ik})$$

Objective Function

- *Objective function* is weighted with (a function of) X_{ij}

$$\arg\min_{w_i, w_j, \ldots} \sum_{i,j=1}^{V} f(X_{ij}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log(1 + X_{ij}) \right)^2$$

- Properties of weighting function $f(X_{ij})$
  - $f(0) \to 0$
  - $f(X_{ij})$ is non-decreasing
  - $f(X_{ij})$ should not increase heavily for large values of x
- Following function is used

$$f(X_{ij}) = \begin{cases} \left(\frac{X_{ij}}{X_{ij}^{max}} \right)^{\alpha} & \text{if } X_{ij} < X_{ij}^{max} \\ 1 & \text{otherwise} \end{cases}$$

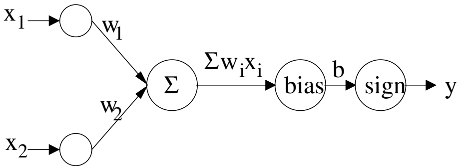
  - $\alpha = 3/4$
  - $X_{ij}^{max} = 100$

Discussion

- Context window of size ± 5
- Dimensionality of 100 or 300
- Captures *local context* and *global pairwise statistics*

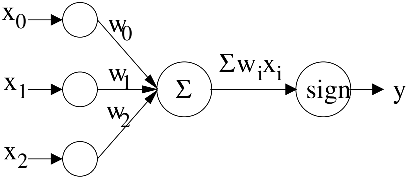
Perceptron

- A perceptron is a simple binary *linear* classifier
- Input attributes $x_1, \ldots, x_n$ are weighted and summed
- A bias b is added as well
- Final class ($y = \pm 1$) is *sign* of the output
- The *sign* node is called the activation function or link/decision/transfer function
- Decision boundary is $\vec{w}.\vec{x} + b = \sum_{i=1}^{n} w_i x_i + b$
- Therefore, sign of $\vec{w}.\vec{x} + b$ predicts the class



Bias

- Why is bias needed?
- Otherwise, hyperplane passes through origin
- Simple trick to model input uniformly: include 1 as x_0 of data
- Decision boundary becomes $w_0 x_0 + \vec{w}.\vec{x} = \sum_{i=0}^{n} w_i x_i$
- Weight on $x_0 = 1$ becomes the constant term, i.e., $w_0 = b$



Examples of Perceptrons

- Different boolean functions
- AND (of x_1 and x_2)
  - $w_1 = w_2 = 1$, $w_0 = -1.5$
- OR (of x_1 and x_2)
  - $w_1 = w_2 = 1$, $w_0 = -0.5$
- NOT (of x_1)
  - $w_1 = -1$, $w_0 = 0.5$
- XOR (of x_1 and x_2)
  - Cannot be done as the two classes are not linearly separable

Learning a Perceptron

- What is new in a linear classifier?
- Training a perceptron, i.e., learning the weights w
- Perceptron learning rule or perceptron training rule

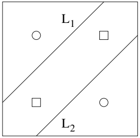
$$w_i = w_i - \eta(\hat{y}_i - y_i)x_i$$

where $\hat{y}_i$ is the predicted value and η is the *learning rate*

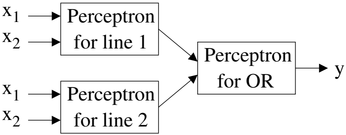
- If $y_i = \hat{y}_i$, there is no change in weight
- If $y_i = +1$ and $\hat{y}_i = -1$, i.e., $\hat{y}_i - y_i < 0$
  - Weights of positive x_i are increased and those of negative x_i are decreased thereby pushing $\hat{y}_i$ towards positive
- If $y_i = -1$ and $\hat{y}_i = +1$, i.e., $\hat{y}_i - y_i > 0$
  - Weights of positive x_i are decreased and those of negative x_i are increased thereby pushing $\hat{y}_i$ towards negative
- If the data *is* linearly separable, a perceptron *will* learn it, i.e., it will converge to the global optimum; otherwise, it may oscillate
- The learning rates η may be modified to give more importance to recent examples
- Weights can be learned through *gradient descent* method as well

Combination of Perceptrons

- A single perceptron can learn only a single hyperplane
- If the data can be separated using two or more hyperplanes, a *combination* of perceptrons can learn it
- Example: XOR

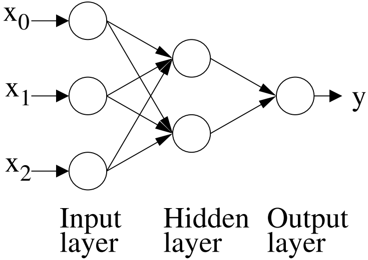


- Each hyperplane can be modeled by a perceptron and the outputs can be combined using OR
  - *Multiple* layers



Artificial Neural Networks

- Artificial neural networks (ANNs) are modeled on the human brain
- Nodes have connections ala neurons in human nervous system
- Nodes are also called neurodes
- Three types of nodes: input layer, *hidden* layer, output layer



- ANN with one hidden layer is considered as *two-layered*
  - Input layer is not counted
- Can have multiple hidden layers
- Learning through ANNs is also called connectionist learning

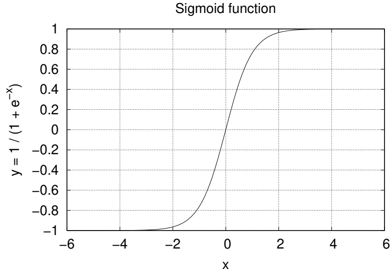
Connections

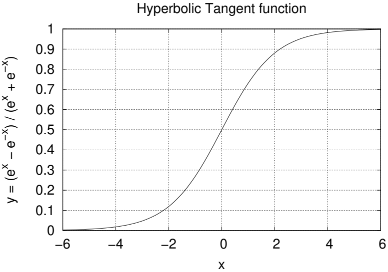
- ANNs are of many types depending on the connections
- The most common are multilayer feed-forward networks
  - Connections are directed from one layer *only* to the next
  - There are *no* back edges or same-layer connections
- In recurrent networks, there can be back edges or same-layer connections
- Fully connected, i.e., each node in one layer is connected to every node in the next layer
- Training is learning the weights on each of these edges
- Akin to layers of perceptrons
- Activation functions in the nodes are *not* linear
  - Combination of linear functions can only learn linear separators
- Activation function is sigmoid (logistic) or hyperbolic tangent

Sigmoid and Hyperbolic Tangent Functions

- If input of a node is x, then output y is

$$y = \sigma(x) = \frac{1}{1 + e^{-x}} \qquad y = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$





- Approximates the step (or sign) function
- Continuous and differentiable
- Output constrained to $(0, 1)$ or $(-1, +1)$
- Scaled versions of each other
- Also called squashing functions

Nodes and Weights

- Output of a node is the sigmoid of the weighted sum of its inputs
- Inputs, in turn, are outputs of previous layers
- Inputs are normalized to $(0, 1)$
- Outputs are already constrained to $(0, 1)$
- Weight from a node i to node j is w_{ij}
- Output from node i is O_i
- Input to node j is I_j

$$I_j = \sum_{\forall i} w_{ij} O_i$$

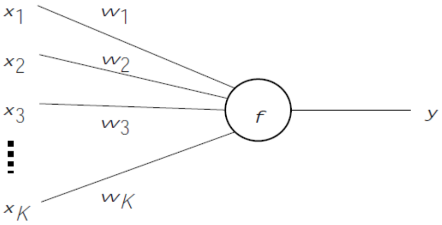
- Output from node j is O_j

$$O_j = \sigma(I_j) = \frac{1}{1 + e^{-I_j}}$$

Training an ANN

- Training an ANN requires
  - Designing the topology: how many layers and many nodes in each layer
  - Learning the weights of the connections
- Designing the topology requires either extensive domain knowledge or simply try-and-test
- The final outputs are some non-linear functions of inputs
- Hence, can be trained using gradient descent
- However, it is too complex and slow
- Weights are updated through the backpropagation algorithm

Backpropagation



- Main idea
  - Start with arbitrary weights
  - Propagate forward the values
  - Propagate backward the errors
  - Update the weights using gradient descent

Backpropagation

- Output, using sigmoid function $\sigma(u) = \frac{1}{1+e^{-u}}$, is

$$y = \sigma(u) = \sigma\left(\sum_{\forall x_i} w_i.x_i\right)$$

- Derivatives of activation functions

$$\text{Sigmoid: } \frac{d\sigma(u)}{du} = \sigma(u)\sigma(-u) = \sigma(u)(1-\sigma(u))$$

$$\text{Tanh: } \frac{d(tanh(u))}{d(u)} = 1 - tanh^2(u)$$

$$\text{Linear: } \frac{d(ReLU(u))}{d(u)} = \begin{cases} 0 \text{ when } u < 0 \\ 1 \text{ when } u \geq 0 \end{cases}$$

Updating Weights

- If true output is t, error is squared error

$$E = \frac{1}{2}(t - y)^2$$

- *Stochastic gradient descent*
- Error function with respect to a single weight w_i

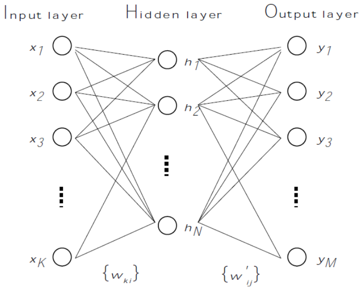
$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y}.\frac{\partial y}{\partial u}.\frac{\partial u}{\partial w_i} = [(y - t)].[y(1 - y)].[x_i]$$

- Therefore, update equation is

$$w_i^{(new)} = w_i^{(old)} - \eta.(y - t).y(1 - y).x_i$$

- Learning rate or *momentum* η controls the speed of training
- Can be continued till error is below a threshold

Backpropagation for Multiple Outputs



$$h_i = \sigma(u_i) = \sigma\left(\sum_{k=1}^{K} w_{ki}.x_{ki}\right)$$

$$y_j = \sigma(u'_j) = \sigma\left(\sum_{i=1}^{N} w'_{ij}.x_{ij}\right)$$

$$E(\vec{x}, \vec{t}, W, W') = \frac{1}{2}\sum_{j=1}^{M}(y_j - t_j)^2$$

Representational Power of an ANN

- Boolean functions
  - Can approximate *any* boolean function with one layer of hidden nodes
  - Number of hidden nodes may be equal to exponential factor of number of boolean variables
  - Output layer wired through AND or OR
- Continuous functions
  - Can approximate *any* continuous function with one layer of hidden nodes up to any arbitrary error factor
  - Due to properties of sigmoid/tanh functions
  - Number of hidden nodes may be large
- Arbitrary functions
  - Can approximate *any* arbitrary function with *two* layers of hidden nodes up to any arbitrary error factor
  - Due to properties of sigmoid/tanh functions
  - Number of hidden nodes may be large

Discussion

- Determining number of nodes and layers is problematic
- Universal approximators
  - Sigmoid and hyperbolic tangent functions
- Gradient descent converges to local minimum only
- Generally, requires large training data
- Very slow to train
- Can be easily parallelized
- Notoriously non-explainable

Term Embedding

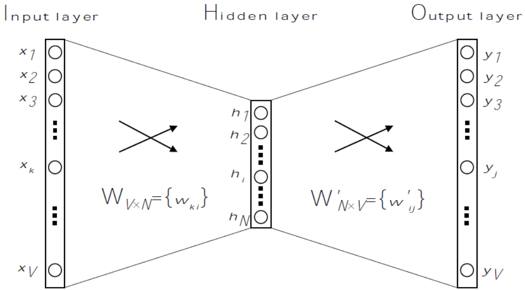
- Aim is to learn an embedding vector for each term that can *predict another term* in its context
- Thus, modeled as a *classification* problem
- ANNs used: Word2Vec model
- If vocabulary size is V, this is a V-class classification problem
- Input is a term *vector*
- Dimensionality is V
- One-hot vector
  - Only the specific term is 1, rest are all 0
- Output layer consists of V nodes
- Only one output vector of dimensionality V
- Softmax used for classification
- Probability of a particular dimension i with value f_i is

$$p(o_i) = \frac{exp(f_i)}{\sum_{\forall i} exp(f_i)}$$

- Choose the word w whose probability $p(w)$ is the *highest*

One-Word Context

- *One* word per context is used to predict *one* target word
  - Subhas : ? Bose



- Only one hidden layer of size N
- Fully connected feed-forward architecture
- Number of weights is $V \cdot N + N \cdot V$
- Given a one-hot encoded vector $\vec{x}_I$ for input w_I, the output is $\vec{y}_O$ predicting the word w_O

Layers

- Weight matrix W of size $V \cdot N$ from input to hidden layer
- Row k represents vector for word k: v_k^T
- For input vector x of word k

$$h = W^T x = W^T I_{(k),\cdot} = v_{w_I}^T$$

- v_{w_I} is the vector representation of input word W_I in N dimensions
- Essentially, the hidden layer is a *linear* copy
- Different weight matrix W' of size $N \cdot V$ from hidden to output layer
- Score for each word w_j is

$$u_j = {v'}_{w_j}^T h = {v'}_{w_j}^T v_{w_I}$$

- v'_{w_j} is the j-th column of W'

Objective

- Using softmax, a multinomial distribution over all predicted words w_j given an input word w_I is defined
- *Log-linear* classification model
- Probability of target word being w_j given an input context word w_I is

$$p(w_j|w_I) = y_j = \frac{exp(u_j)}{\sum_{j'=1}^{V} exp(u_{j'})} = \frac{exp({v'}_{w_j}^{T} v_{w_I})}{\sum_{j'=1}^{V} exp({v'}_{w_{j'}}^{T} v_{w_I})}$$

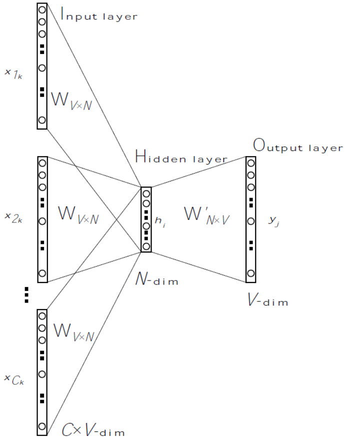
- v_w is the *input vector* of word w
- ${v'}_w$ is the *output vector* of word w
- Training objective is to *maximize* the probabilities

$$\max p(w_{j*}|w_I) = \max y_{j*} = \max \log y_{j*} = u_{j*} - \log \sum_{j'=1}^{V} exp(u_{j'}) = -E$$

- Which is the word vector representation?
- *Hidden layer* of dimensionality N
- Training is through backpropagation

Continuous Bag-of-Words (CBOW) Model

- CBOW: *Multiple context words* to predict a *single target word*
  - Tendulkar, Dravid, Laxman : ? Ganguly
  - Sachin, Rahul : ? DevVarman



- Bag-of-words in a local context window that continuously changes

Model

- Instead of just a copy, hidden node is average of C context words

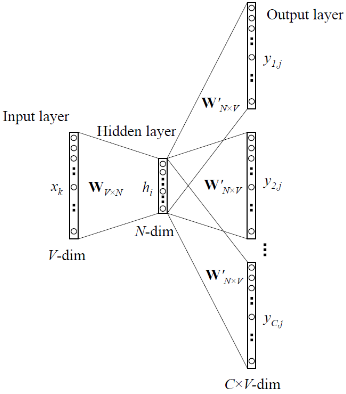
$$\vec{h} = \frac{1}{C} W^T (x_1 + x_2 + \cdots + x_C) = \frac{1}{C} (v_{w_1} + v_{w_2} + \cdots + v_{w_C})^T$$

- Error

$$E = -\log p(w_{j*} | w_{I,1}, \ldots, w_{I,C}) = -u_{j*} + \log \sum_{j'=1}^{V} exp(u'_j)$$

Skip-Gram Model

- Skip-gram: *Single context word* to predict *multiple target words*
  - DevVarman : ? , ? Sachin, Rahul



- Output layer is C target words of dimensionality V

Model

- Output layer weights are shared, i.e., they are the same matrix W'
- Probability of c^{th} target word being $w_{c,j}$ given an input context word w_I is

$$p(w_{c,j}|w_I) = y_{c,j} = \frac{exp(u_{c,j})}{\sum_{j'=1}^{V} exp(u_{j'})} = \frac{exp({v'}_{w_j}^{T} v_{w_I})}{\sum_{j'=1}^{V} exp({v'}_{w_{j'}}^{T} v_{w_I})}$$

- Error

$$E = -\log p(w_{j*,1}, \ldots, w_{j*,c}|w_I) = -\log \prod_{c=1}^{C} \frac{exp(u_{c,j*_c})}{\sum_{j'=1}^{V} exp(u_{j'})}$$

Word2Vec Discussion

- Context window size used is 4 words up and down
- Note that, k is chosen randomly within 4
  - This is why it is called "continuous" bag-of-words
- Number of context words, C, is within 5
- Rare words are discarded
  - This also effectively increases context window size
- Log-linear model
  - This makes training faster
- Input word and context word are treated differently
  - No good reason except mathematical convenience
- Skip-gram gives better semantic results
- *Corpus* is more important than method
- Why does just context?
  - No definitive or satisfying answer

Morphologically Richer Languages

- Morphologically richer languages have the same internal root or structure for a large number of words
- Further, it requires a prohibitively large corpus to train for all such variants of a word
- Examples: Indian languages, Finnish, Turkish, and even European languages such as French, Spanish, German, Russian
- In Indian languages, compound words are a problem too
- Simple corpus-based word vector embeddings may not work
- Byte-pair encodings, etc. alleviate the problem
- FastText uses character n-grams
- Originally called SISG (Subword Information Skip Gram)

FastText

- A word vector representation is *sum* of character n-gram vectors and the word itself
- Two limits for n-gram length: minimum and maximum
  - Generally, 3 and 6
- Example word: "India"
- If limits are 2 and 3, then embeddings are of "In", "Ind", "ndi", "dia", "ia", and "India"
- If G_w is the set of n-grams for a word w, then

$$s(w, c) = \sum_{\forall g \in G_w} z_g^T v_c$$

- FastText can handle *out-of-vocabulary* words
  - Glove and Word2Vec cannot!
- Uses as many common n-grams as it can from the new word