

Sumarizzing

Programming Club

Introduction to NLP(2/3)

Content to be covered...

- 1 Vector Semantics and Embeddings
- 2 Neural Networks for NLP
- 3 RNNs and LSTMs

Content to be covered..



1 Vector Semantics and Embeddings

2 Neural Networks for NLP

3 RNNs and LSTMs

Vector Semantics and Embeddings

Words that occur in similar contexts tend to have similar meanings. This link between similarity in how words are distributed and similarity in what they mean is called the **distributional hypothesis**.

- We intend to develop models that give us these word meanings.
- Try learning about the following terms: **word sense disambiguation**, **propositional meaning**.
- Word-similarity is how similar the word meanings are.
Word-relatedness is more than word-similarity. It also involves context and the word it surrounds.
- A **semantic field** is a set of words which cover a particular semantic domain and bear structured relations with each other. For example, words might be related by being in the semantic field of hospitals (surgeon, scalpel, nurse, anesthetic, hospital), restaurants (waiter, menu, plate, food, chef), or houses (door, roof, kitchen, family, bed). Semantic fields are also related to topic models, like **Latent Dirichlet Allocation (LDA)**.
- Latent Dirichlet Allocation is implemented in scikit-learn. Explore it.  

Vector semantics is the standard way to represent word meaning in NLP, helping us model many of the aspects of word meaning.

- The idea of vector semantics is to represent a word as a point in a multidimensional semantic space that is derived from the distribution of word neighbours.
- Vectors for representing words are called **embeddings**. However, it more generally applied to dense-vector representations like **word2vec** rather than sparse **tf-idf** or **PPMI**.

- **Vector and documents:** in the **term-document** matrix each row represents a word on the vocabulary and each column represents a document and each cell represents the number of times a word appeared in that document.
- Term-document matrices were defined as a means of finding similar documents for the task of document information retrieval. Two documents that are similar will tend to have similar words, and if two documents have similar words their column vectors will tend to be similar.
- In a very similar fashion **term-term** matrix or the **term-context** matrix is also defined. They are of the size $|V| \times |V|$, where V is the size of the vocabulary.
- How is similarity measured: Kernels and cosines.

TF-IDF: Weighing terms in a vector

- Raw frequency is very skewed and not very discriminative.
- There are two terms. First is the **term frequency (tf)**. The frequency of the word t in the document d . This is generally viewed on a logarithmic scale. Thus, $tf_{t,d} = \log_{10}(\text{count}(t, d) + 1)$.
- The second term is used to give higher weight to those terms that occur only in a few documents. Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection. There comes the **inverse-document frequency** $idf_t = \log_{10}\left(\frac{N}{df_t}\right)$, where df_t is the number of documents with the term t .
- The net weight given is $w_t = tf_{t,d}.idf_t$. Thus the term-document matrix or the term-context matrix is modified.

PMI: Pointwise Mutual Information

Try reading about this on your own :)

- The vectors we have seen thus far are long and sparse. There are much powerful word representations: **embeddings**, which are dense vectors, of dimension d . The dimension d do not have a clear interpretation.
- A method to compute these embeddings is: **skip-gram with negative sampling**.
- The word2vec implementations produce static embeddings, meaning the method learns one fixed embedding for each word in the vocabulary.
- There is also something called a contextual embedding (discussed after transformers).

- The intuition behind skip-gram model is that:
 - Treat then target word and a neighbouring context words as positive samples.
 - Randomly sample other words in the lexicon to get the negative samples.
 - Train a logistic regression model to train a classifier to distinguish those cases.
 - Use the learned weights as the embeddings.
- Consider, $[..., c1, c2, w, c3, c4, ...]$ as a sequence of words with window size = 2. Our goal is to learn a model that returns a probability whether c is a correct context word given a tuple (w, c) to the model. The model's intuition is that the similar the vectors are, the similar the words would be. Thus $P(+|w, c) = \sigma(c.w)$.
- Skip gram further makes an assumption, that all context words are independent, thus:

$$P(+|w, c_{1:L}) = \prod_{i=1}^L \sigma(c_i.w)$$

Learning skip-gram embeddings

- The Skip-gram actually learns two embeddings, one for a word as a target and other for a word as a context. The two matrices \mathbf{W} and \mathbf{C} have to be learnt, for each of $|V|$ words.
- Given a vocabulary of size N , we assign random vectors to each word of dimension d , and slowly shift these embeddings to words that occur nearby. In a standard implementation, for each positive sample, a k negative samples are used.
- We then use our cross entropy loss functions to calculate the loss and derivatives:

$$\begin{aligned} L_{CE} &= -\log \left[P(+|w, c_{pos}) \cdot \prod_{i=1}^k P(-|w, c_{neg}) \right] \\ &= - \left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^k \log \sigma(-c_{neg} \cdot w) \right] \end{aligned}$$

- Calculate the derivatives with respect to c_{pos} , c_{neg} , w , and apply gradient descent.

Further readings on semantic associations:

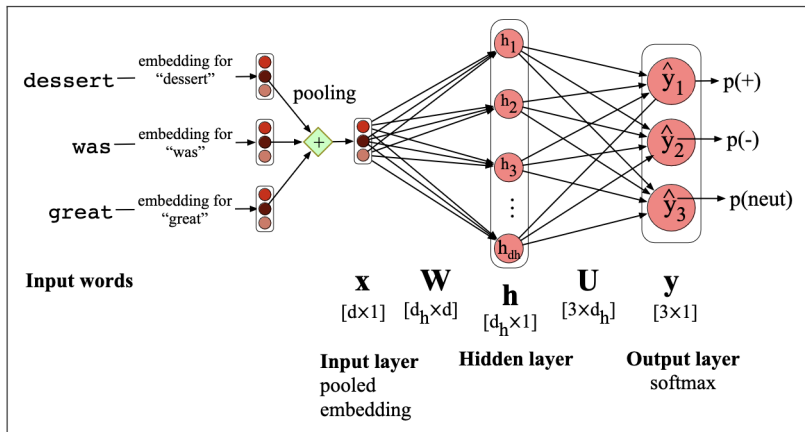
- first-order co-occurrence (syntagmatic association): Schutze and Pedersen, 1993
- second-order co-occurrence (paradigmatic association)
- Historical semantics.

Content to be covered..

- 1 Vector Semantics and Embeddings
- 2 Neural Networks for NLP
- 3 RNNs and LSTMs

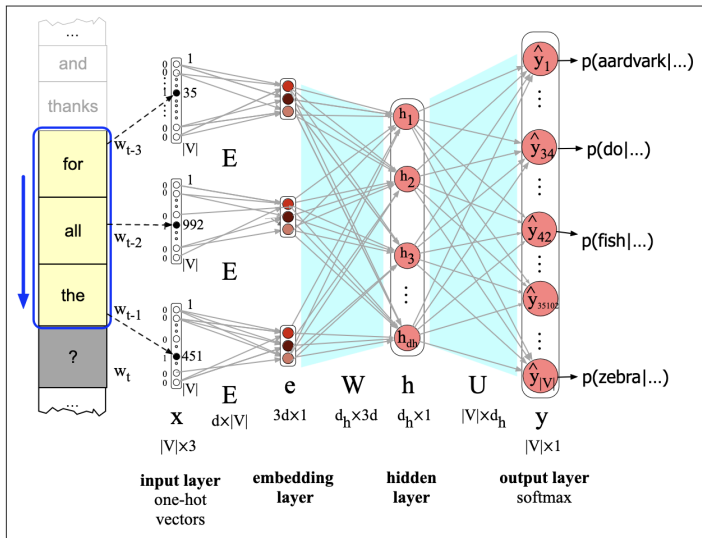
Feedforward neural networks for NLP: Classification

- Pool our input embeddings and use the neural network to produce output tags for sentiment analysis.



Feedforward neural networks for NLP: Language Model

- Derive a probability distribution over the corpus given a window of context.

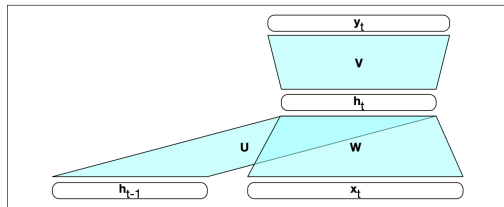


Content to be covered..

- 1 Vector Semantics and Embeddings
- 2 Neural Networks for NLP
- 3 RNNs and LSTMs

Recurrent Neural Networks

- RNN's are neural networks that contain a cycle within its network connections. This adds a temporal dimension to this structure. We need to define new weights **U** (say) to configure previous computations in our previous hidden layers.



$$h_t = g(Uh_{t-1} + Wx_t)$$

$$y_t = f(Vh_t)$$

$$= \text{softmax}(Vh_t)$$

RNNs as Language Model

Assume a input sequence $X = [x_1; x_2; \dots; x_N]$ consists of one-hot vectors of size $|V| \times 1$.

We first retrieve the word embeddings using a E embedding matrix.

$$e_t = E.x_t$$

$$h_t = g(U.h_{t-1} + W.e_t)$$

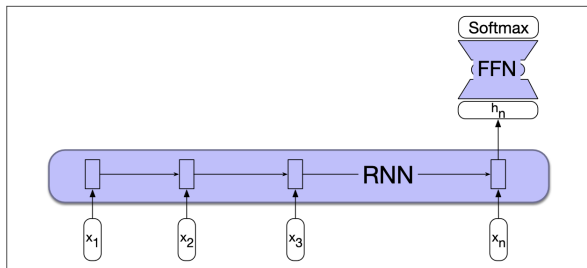
$$y_t = \text{softmax}(V.h_t)$$

$$P[w_{t+1} = i | w_1, w_2, \dots, w_t] = y_t[i]$$

We further define the probability of a sequence in a similar way.

RNNs for Sequence Classification

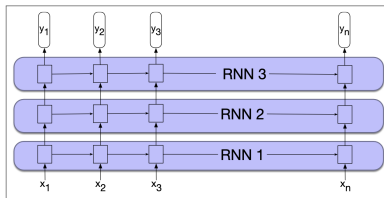
Another use of RNNs is to classify entire sequences rather than the tokens within them. This is the set of tasks commonly called text classification, like sentiment analysis or spam detection, in which we classify a text into two or three classes.



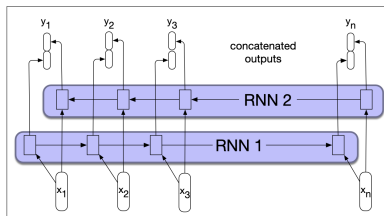
One way is to use the last token generation h_n generated and feed the representation to feed forward network, finally via a softmax to generate probabilities for the output labels. We could alternatively pool the hidden state representations.

Stacked and Bidirectional RNN architecture

Stacked RNNs consist of multiple networks where the output of one layer serves as the input to a subsequent layer.



In the left-to-right RNNs, the hidden state at a given time t represents everything the network knows about the sequence up to that point, $h_t^f = RNN_{forward}(x_1, x_2, \dots, x_t)$. To take advantage of context to the right of the current input, we can train an RNN on a reversed input sequence, $h_t^b = RNN_{backward}(x_t, \dots, x_n)$



- Despite having access to the entire preceding sequence, the information encoded in hidden states tends to be fairly local, more relevant to the most recent parts of the input sequence and recent decisions. Yet distant information is critical to many language applications. Consider the following example in the context of language modeling.
- One of the reasons is during the backward pass of training, the hidden layers are subject to repeated multiplications, as determined by the length of the sequence. A frequent result of this process is that the gradients are called the vanishing gradients problem.
- LSTMs divide the context management problem into two subproblems: removing information no longer needed from the context, and adding information likely to be needed for later decision making.

LSTMs add an explicit context layer to the architecture and through the use of specialized neural units that make use of gates to control the flow of information into and out of the units that comprise the network layers.

[A video on LSTM by StatQuest.](#)

The choice of the sigmoid as the activation function arises from its tendency to push its outputs to either 0 or 1. Combining this with a pointwise multiplication has an effect similar to that of a binary mask.

- The first gate is the **forget gate**. The purpose of this gate is to delete information from the context that is no longer needed.
- The forget gate computes a weighted sum of the previous state's hidden layer and the current input and passes that through a sigmoid. This mask is then multiplied element-wise by the context vector to remove the information from context that is no longer required. Element-wise multiplication of two vectors (represented by \odot)

$$f_t = \sigma(U_f \cdot h_{t-1} + W_f x_t)$$

$$k_t = c_{t-1} \odot f_t$$

- The next task is to compute the actual information we need to extract from the previous hidden state and current inputs—the same basic computation used for all our recurrent networks.

$$g_t = \tanh(U_g \cdot h_{t-1} + W_g \cdot x_t)$$

- Next, to generate a mask for the **add gate** to select information to add to the current context.

$$i_t = \sigma(U_i \cdot h_{t-1} + W_i \cdot x_t)$$

$$j_t = g_t \odot i_t$$

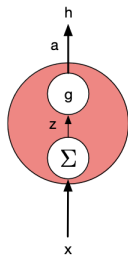
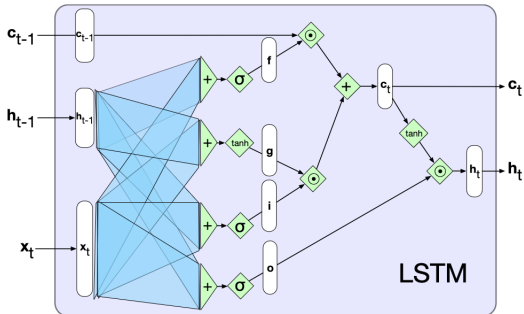
This is added to the modified context vector to get the new context vector.

$$c_t = j_t + k_t$$

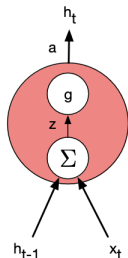
- The final gate is the **output gate** which is used to decide what information is required for the current hidden state (as opposed to what information needs to be preserved for future decisions).

$$o_t = \sigma(U_o/h_{t-1} + W_o.x_t)$$

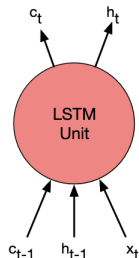
$$h_t = o_t \odot \tanh(c_t)$$



(a)



(b)



(c)

What we'll cover next:

- Encoder-decoder architecture
- Attention mechanism
- Transformers

Thank you!