2025-09-01T04-15-20
Tags: DLS

DLS C5 week 2

Word representations

Papers mentioned:

- Visualizing Data using t-SNE by Laurens van der Maaten and Geoff Hinton.

Words are represented as one-hot vectors.

- If “man” is word no. 5391 (O_{5391}): it is a vector with 1 as position 5391 and 0 everywhere.

One-hot vectors treat each word as an independent object.

- It does not capture relationships or similarities across words.

Mathematical reason on why one-hot vectors are independent:

- The inner product between any two different one-hot vector is zero.
- Distance between any two one-hot vector is the same.

Instead of a one-hot representation, learn a featurized representation where each word can be represented by a set of features.

For example: 300 features means a 300-dimensional vector.

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Age	0.03	0.02	0.7	0.69	0.03	-0.02
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Food	0.04	0.01	0.02	0.01	0.95	0.97
Notation	e_{5391}	e_{9853}	e_{4914}	e_{7157}	e_{456}	e_{6257}

Using featurized vectors allows:

- generalization across similar words.
- captures semantic relationships.

It is common practice to reduce a high dimensional (300-d) vector to 2D for visualization. A common algorithm for doing this is the *t-SNE* algorithm.

Insights during visualization:

- related words cluster together.

Emebeddings are mapping of words into a high-dimensional vector space—where each word is a point in that space.

Using word embeddings

Using word embeddings procedure:

1. Learn word embeddings from a large text corpus (which are 1-100B words).
    - An alternative is to download pre-trained embeddings online.
2. Transfer the embedding to a new task with a smaller train set (i.e., 100k words).
3. (Optional) finetune the word embedding with new data.
    - If the label data for step 2 is small, don't finetune.

Useful	Less useful
Named entity recognition	Language modeling
Text summarization	Machine translation
Co-reference	
Parsing	

Word embedding vs. face recognition encoding:

- Both are fairly similar.
- In face recognition:
    - Encoding refers to vectors $f(x(i))$ and $f(x(j))$.
    - Train a neural network to take face picture as input.
    - Have the neural network compute an encoding for the new picture.
    - Used with unlimited pictures.
- In word embeddings:

- Have a fixed vocabulary (e.g., 10000 words).
- Learn a fixed encoding (embedding) for each word in the vocabulary.
- Used with a fixed vocabulary.

Properties of word embeddings

Papers mentioned:

- Linguistic Regularities in Continuous Space Word Representations by Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig.

Word embeddings can be used for naalogy reasoning.

- For example: man is to woman as king is to queen s.t. $e_{\text{man}} - e_{\text{woman}} \approx e_{\text{king}} - e_{\text{queen}}$.

Carryying out an analogy reasoning:

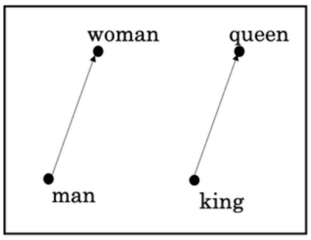
- Find a word that $e_{\text{man}} - e_{\text{woman}} \approx e_{\text{king}} - e_{?}$
- Find word w using $\text{argmax}_{\text{w}}\ \text{sim}(e_w, e_{\text{king}} - e_{\text{man}} + e_{\text{woman}})$
- Use cosine similarity to calculate the similiarity.

Know that:

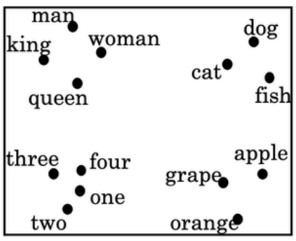
$$\text{cosine similarity}(u, v) = \frac{u^\intercal \cdot v}{||u||_2||v||_2}$$

- $u = e_{\text{w}}$
- $v = e_{\text{king}} - e_{\text{man}} + e_{\text{woman}}$

What does t-SNE do? It takes, let's say, a 300-D data and maps it in a very non-linear way into a 2D space.



300 D



t-SNE

Embedding matrix

When implementing an algorithm to learn a word embedding, what ends up is an embedding matrix.

- Suppose we're using 10000 words as our vocabulary plus token.
- The algo should create a matrix E of shape $(300, 10000)$ if we're extracting 300 features.

- If O_{6237} is the one-hot encoding for the word *orange* of shape (10000, 1), then `np.dot(E, O_6327) = e_6257` of shape (300, 1).
- Generally `np.dot(E, O_j) = e_j`.

We initialize E randomly and try to learn all params of this matrix.

It's not efficient to use dot multiplication when extracting embeddings of a specific word.

- Instead, use slicing to slice a specific column.
- In Keras, there is an embedding layer that extracts this column with no multiplication.

Learning word embeddings

Papers mentioned:

- A Neural Probabilistic Language Model by Yoshua Bengio, Rejean Ducharme, Pascals Vincent, and Christian Jauvin.

Say, we want to build a language model that can predict the next word.

Using a neural network to learn the language model:

- Get e_j using `np.dot(E, o_j)`.
  - E times the one-hot vector o_j gives the embedding vector.
  - E — the embedding matrix.
  - o_j — one-hot vector for word j.
- The neural network consists of:
  - Hidden layers with params W_1 and b_1.
  - Softmax output layer with params W_2 and b_2.
- The input dimension is $(300 \times 6, 1)$ if the window size is 6.
  - If $\text{window size} = n$, then that means n previous words.
- Optimize E and the network parameters during training.
  - The objective is to maximize the likelihood of predicting the next word given the context (previous words).

Example

Take into account the sentence: *I want a glass of orange juice to go along with my cereal.*

- To learn the word *juice*, there are several choices for context:
  - Last 4 words.
  - 4 words on the left and on the right.
  - Last 1 word.
  - Nearby 1 word.
- Scientific findings tell that it's natural to use *the last few words* as a context when building a language model.
- Use all of the context when learning a word embedding.

Word2Vec

Papers mentioned:

- Efficient Estimation of Word Representations in Vector Space by Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean.

Skip-gram model

The *skip-gram* model:

1. Instead of having context always be the *last four words*, pick a word to be the *context word*.
2. Pick another word within some window and choose that to be the *target word*.
3. Set up a supervised learning problem given the context word.
   - The model predicts a randomly chosen word within a window of the input context word.
   - This is not easy as for ± 10 words, there's a lot of different words.
   - The goal of setting up the supervised learning problem is to learn a good word embedding.

Say we use the previous example: *I want a glass of orange juice to go along with my cereal*.

Context	Target	How far
orange	juice	+1
orange	glass	-2
orange	my	+6

Model details:

$$o_c \to E \to e_c \to O(\text{softmax}) \to \hat{y}$$

1. Given a vocabulary size of 10000.
2. We want to learn a mapping from some context c to target t.
3. Represent the context word with a one-hot vector o_c.
4. Multiply embedding matrix E by o_c to get $e_c = Eo_c$.
5. Feed e_c to a softmax unit to get $\hat{y}$.

Softmax unit: $P(t|c) = \dfrac{\exp(\theta_t^\intercal e_c)}{\sum_{j=1}^{10000} \exp(\theta_j^\intercal e_c)}$

- θ_t — parameter associated with output t.
- Bias term is omitted.

Suppose you have a 10000 word vocabulary, and are learning 500-dimensional word embeddings.

- θ_t and e_c are both 500 dimensional vectors.

Loss function: $\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{10000} y_i \log \hat{y}_i$

- y is a 10,000-D one-hot vector representing the target word.
  - e.g., target word *juice* of ID 4834 will have a one-hot vector where $y_{4834} = 1$ and 0 for others.
- $\hat{y}$ is a 10,000-D vector with probabilities for all 10,000 possible targets words.

Problem with softmax:

- Every softmax step requires summing over the 10,000 (or even more) words in the vocabulary. This is computationally slow and expensive.

One solution to the softmax problem: use a *Hierarchical softmax classifier*.

- Instead of a flat softmax, use a *binary tree* of classifiers.
- For a vocabulary of 10,000 words:
  - First classifier — is the target word in the first or last 5,000?
  - Subsequent classifiers — narrows down until a leaf node corresponding to the target word is reached.
- Reduces complexity from $O(W)$ to $O(\log W)$.
- In practice, the classifier a heuristic is implemented where:
  - Common words are placed on top.
  - While less common words are buried much deeper in the tree.

CBOW

The other version of Word2Vec involves the CBOW (continuous bag of words) model.

- Takes the surrounding contexts from middle word.
- Use the surrounding words to try to predict the middle word.

Negative sampling

Papers mentioned:

- Distributed Representations of Words and Phrases and their Compositionality by Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeff Dean.

Negative sampling allows similar objectives to the skip-gram model but with a lower computational cost.

Positive and negative examples:

1. Generate a positive example.
   1. Sample a context word and a target word.
   2. Associated the pair with a label of 1.
2. Generate negative examples.
   - Take the context word and pick another random word k times.
   - Choose large values of k for smaller data sets (e.g., 5 to 20).
   - Choose smaller k for large data sets (e.g., 2 to 5).

In this table example: $k = 4$

context	word	target?
orange	juice	1
orange	book	0
orange	the	0
orange	of	0

Compared to the original skip-gram model:

- Instead of training all 10,000 words on every iteration, only train $k+1$.

How to choose the negative examples? There are three methods:

- Sample according to the empirical frequency of words in the corpus.
  - Problem — end up with a high representation of common words like *the*, *of*, *and*, etc.
- Use $p(w) = 1/|V|$ to sample negatives at random.
  - Problem — unrealistic distribution.
- Use $p(w) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10000} f(w_j)^{3/4}}$.
  - $f(w_i)$ is the observed frequency of word w_i.
  - Used by the mentioned paper.

GloVe word vectors

Papers mentioned:

- GloVe: Global Vectors for Word Representation by Jeffrey Pennington, Richard Socher, Chris Manning.

GloVe means Global Vectors for Word Representation. Not as widely-used but popular for its simplicity.

Instead of sampling context-target pairs based on proximity. Glove makes co-occurence counts explicit.

X_{ij} — times j appears in the context of$i$.

- Think of X_{ij} as X_{ct}.
- If the context is always the word immediately before the target word, then X_{ij} is not symmetric.
- Symmetry is dependent on the context:
  - If context is ± 10 words, $X_{ij} = X_{ji}$ (syemmtric).
  - If context is the word immediately before target, it is not symmetric.

GloVe uses a co-occurence matrix as starting point.

GloVe model: $\text{minimize} \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij}) \left(\theta_i^\intercal e_j + b_j + b'_j - \log(X_{ij})\right)^2$

- $\theta_i^\intercal e_j$ plays the role of $\theta_t^\intercal e_c$.
- We want to learn vectors so that their end product is a good predictor of how often the two words occur.

Handling zero co-occurences: if $X_{ij} = 0 \to \log(0) \to = -\infty \to \text{undefined}$

- Add a weighting function $f(X_{ij})$ where $f(X_{ij}) = 0$ when $X_{ij} = 0$.
  - Down-weights very frequent words.
  - Avoids giving miniscule weight to rare words.
  - There are various heuristics in the GloVe paper for the function.

Unlike skip-gram: θ_i and e_j play symmetric roles.

- After training: the final embedding for word w is $(\theta_w + e_w)/2$.

GloVe works because

- It's based on co-occurence statistics.
- Encodes global distributional information.

The original motivation is that each dimension is a feature. But in reality, it's not guaranteed that axes align with human-interpretable features.

- Any invertible linear transformation (e.g., matrix A) can rotate embedding space without changing performance: $\theta_i^\top e_j = (A\theta_i)^\top (A^{-1} e_j)$
- Thus, dimensions are not interpretable individually.
- Each dimension is usually a mixture of semantic properties.

However, even after arbitrary transformations:

- Word analogies still work because vector differences remain meaningful.
- Therefore, GloVe learns useful embeddings despite the lack of interpretability of single dimensions.

GloVe pipeline procedure:

1. Build a co-occurence matrix X_{ij}.
2. $\text{minimize } \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij}) \big(\theta_i^\top e_j + b_j + b_j' - \log(X_{ij})\big)^2$
3. Average θ and e using $(\theta_w + e_w)/2$ to obtain the final embeddings.

Concluding insights on word embeddings:

- On first try, download a pre-trained model.
- Once enough data is available, try implementing the discussed algorithms.
- Most practitioners load a pre-trained set of embeddings because word embeddings are computationally expensive to train.

Sentiment classification

Sentiment classification is the process of finding if a text has a positive or negative review.

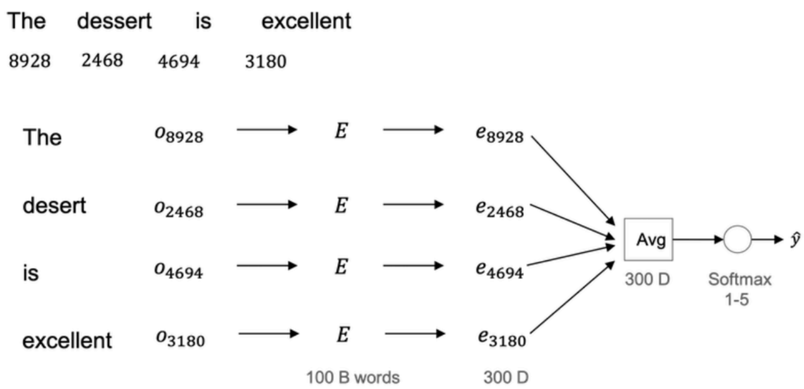
One challenge that sentiment classification faces is the absence of a huge labeled train dataset.

Commmon dataset size varies from 10,000 to 100,000 words.

Example 01

The following example is a simple sentiment classification model.

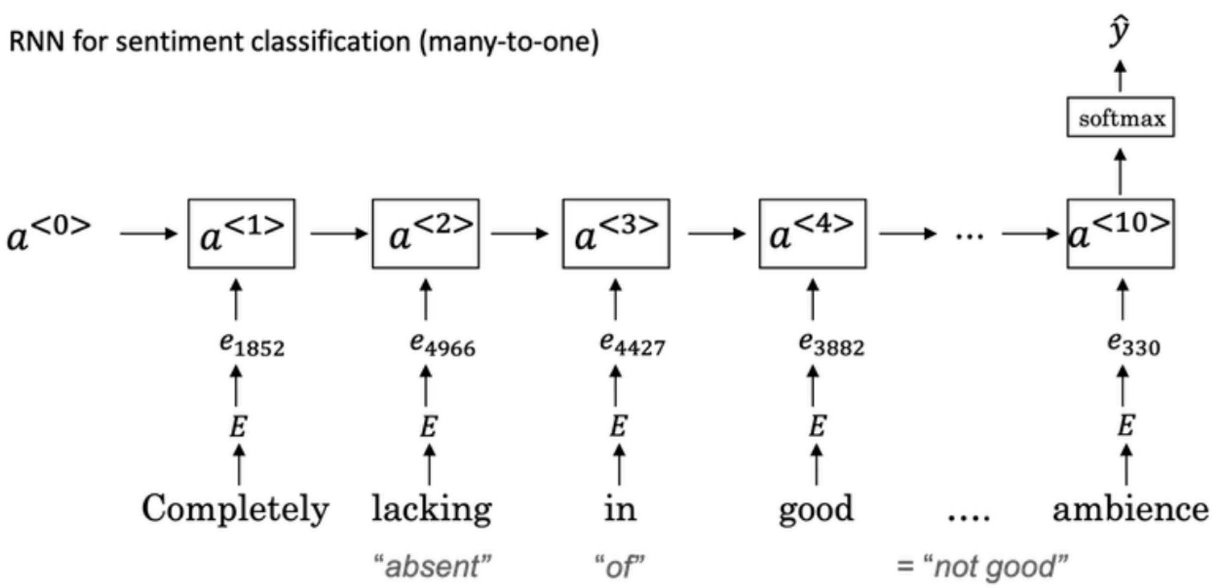
comments	stars
The dessert is excellent.	4
Service was quite slow.	2
Good for a quick meal, but nothing special.	3
Completely lacking in good taste, good service, and good ambience.	1



- E trained on a 100 billion words.
- The number of features in the word embedding is 300.
- Average (or sum) all the feature vectors for every word.

Problems for this model: it ignores word order.

Example 02



Instead of summing (or averaging) all of the word embedding, use a RNN for sentiment classification.

- This generalizes better even if a word is absent from the dataset.

Debiasing word embeddings

Paper:

- Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings by Tolga Bolukbasi, Kai-Wei Chang, James Zou, Venkatesh Saligrama, Adam Kalai.

Word embeddings may have a *social* bias problem.

- This includes gender bias, ethnicity bias, etc.
  - e.g., Man is to Computer Programmer as Women is to Homemaker.
- It is not referring to the bias-variance tradeoff in ML.

The paper shows a way to reduce gender bias in word embeddings:

1. Identify bias direction (e.g., gender)
   - Take a few differences k and average them.
     - $g_1 = e_{\text{he}} - e_{\text{she}}$
     - $g_2 = e_{\text{male}} - e_{\text{female}}$
   - If the original embedding is 300-D
     - Bias direction is a 1-D subspace representing gender.
     - Non-bias vector is a 299-D vector.
2. Neutralize
   - Remove bias from neutral words.
   - For every word that is not definitional, project to get rid of bias.
3. Equalize pairs:
   - Ensure paired words differ only in gender, not in similarity to other neutral words.
   - E.g., Move words like grandmother and grandfather equidistant from gender neutral words like doctor or babysitter.

Which words to neutralize? Train a classifier to do this.

- The classifier will identify definitional words where gender is inherent in meaning.
- Most words are not definitional. Hence, they can be neutralized.

Because equlization pairs are small in number, they are feasible to be hand-picked.