DEEP LEARNING

Word2vec with PyTorch: Implementing the Original P

Covering all the implementation details, skipping high attached.

Olga Chernytska

Sep 29, 2021 • 16 min read

Thoughts and Theory

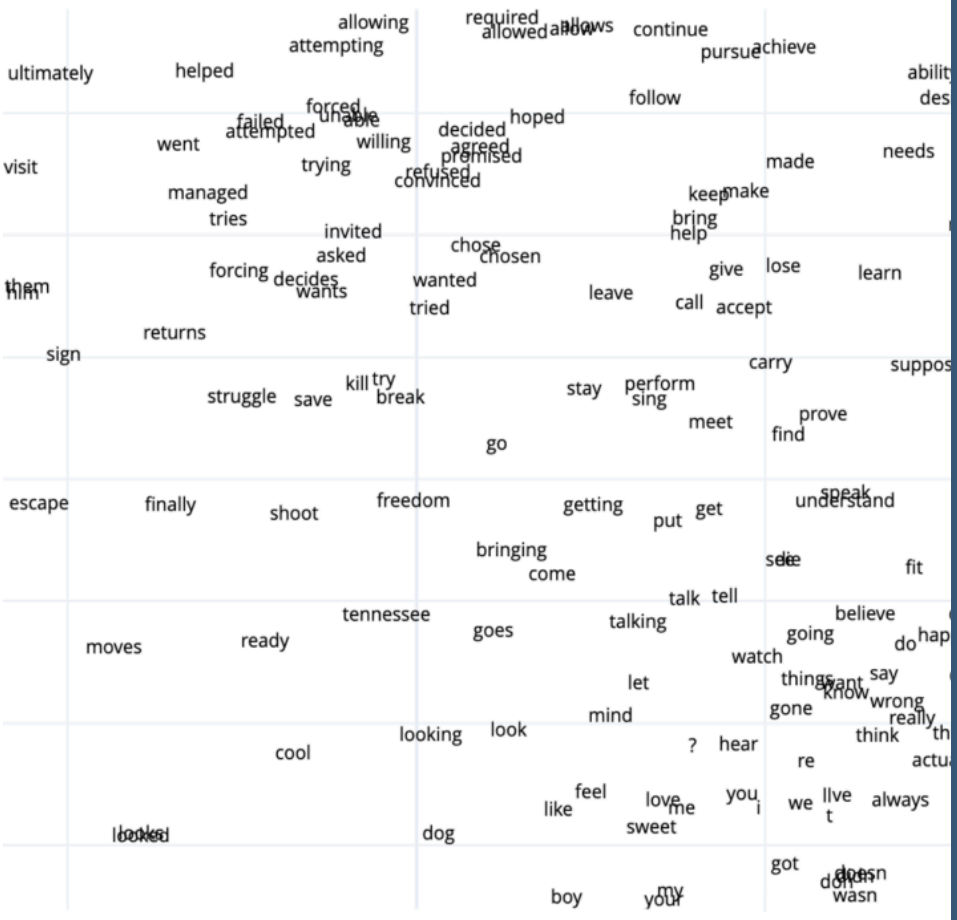


Image by Author

Word Embeddings is the most fundamental concept in Deep Natural Language Processing. And Word2vec is one of the earliest algorithms used to train word embeddings.

In this post, I want to go deeper into the first paper on word2vec – Efficient Estimation of Word Representations in Vector Space (2013), which as of now has 24k citations, and this number is still growing.

Our plan is the following:

- Review model architectures described in the
- Train word2vec model from scratch using Py
- And evaluate the word embeddings that we

I am attaching my Github project with word2vec through it in this post.

Today we are reviewing only the ***first*** paper on w there are several later papers, describing the evo

- Distributed Representations of Words and Ph Compositionality (2013) describes several ex original word2vec to speed up training and i quality.
- Distributed Representations of Sentences an shows how to use the idea behind word2vec and document embeddings. This approach is
- Enriching Word Vectors with Subword Inform introduces even more extensions to word2ve operates on the character level (not words a known as fastText.

I believe, if you understand the first paper, you'll easily catch the ideas described in later papers. So let's go!

Disclosure. Wor2vec is already an old algorithm and there are more recent options (for instance, [BERT] (https://en.wikipedia.org/wiki/BERT(languagemodel))). This post is for those, who have just started their journey into Deep NLP, or for those, who are interested in reading and implementing papers.

Contents

What is word2vec?

Here is my 3-sentence explanation:

1. Word2vec is an approach to create word em
2. Word embedding is a representation of a wo
   vector.

3. Except for word2vec there exist other methods to create word embeddings, such as fastText, GloVe, ELMO, BERT, GPT-2, etc.

If you are not familiar with the concept of word embeddings, below are the links to several great resources. Read through skipping the details but grasping the intuition behind it. And come back to my post for the word2vec details and coding.

- Why do we use word embeddings in NLP by
- The Illustrated Word2vec by Jay Alammar
- An introduction to word embeddings for text Lynn

Better now?

Word embeddings are used literally in every NLP classification, named-entity recognition, questio summarization, etc. Everywhere. Models do not and letters, they understand numbers. That's wh embeddings come in handy.

Model Architecture

Word2vec is based on the idea that a word's me its context. Context is represented as surroundi

Think about it. Assume, you are learning a new l reading a sentence and all the words there are f except one. You've never seen this word before, tell its part of speech, right? And sometimes, ev meaning. That's because the information from s helps you.

For the word2vec model, context is represented as N words before and N words after the current word. N is a hyperparameter. With larger N we can create better embeddings, but at the same time, such a model requires more computational resources. In the original paper, N is 4–5, and in my visualizations below, N is 2.

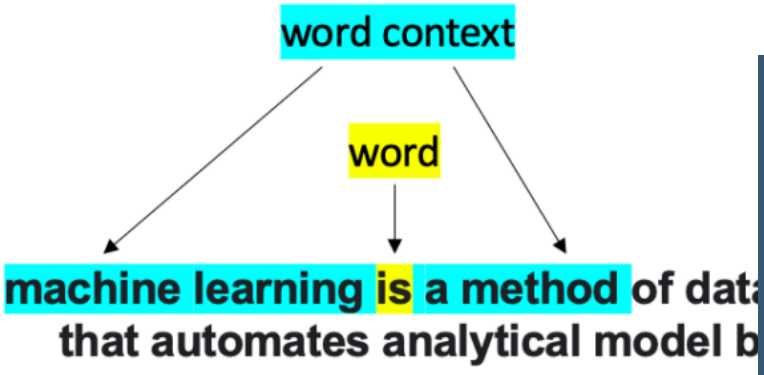


Image 1. A word and its context. Image by Autho

There are two word2vec architectures proposed

- **CBOW** (Continuous Bag-of-Words) – a mode current word based on its context words.
- **Skip-Gram** – a model that predicts context current word.

For instance, the CBOW model takes "machine", "method" as inputs and returns "is" as an output model does the opposite.

Both CBOW and Skip-Gram models are multi-cla models by definition. Detailed visualizations belo clear.

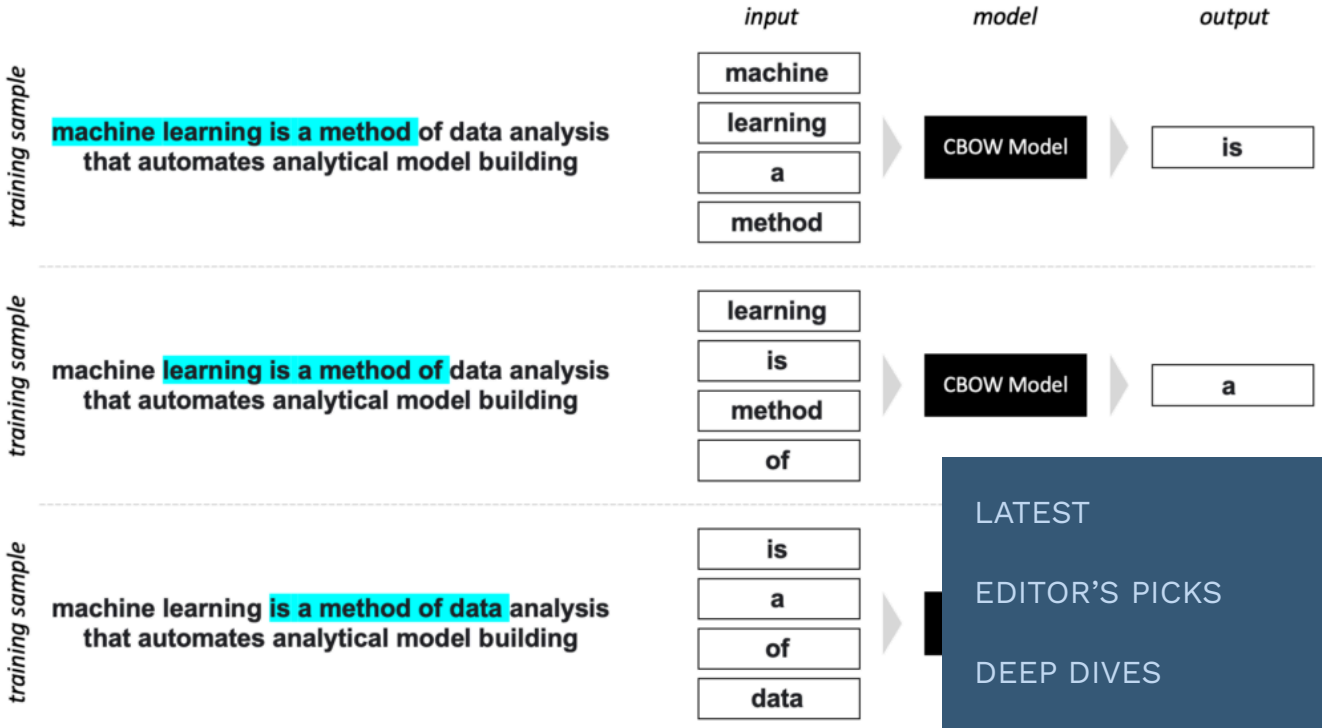


Image 2. CBOW Model: High-level Overview. Image by

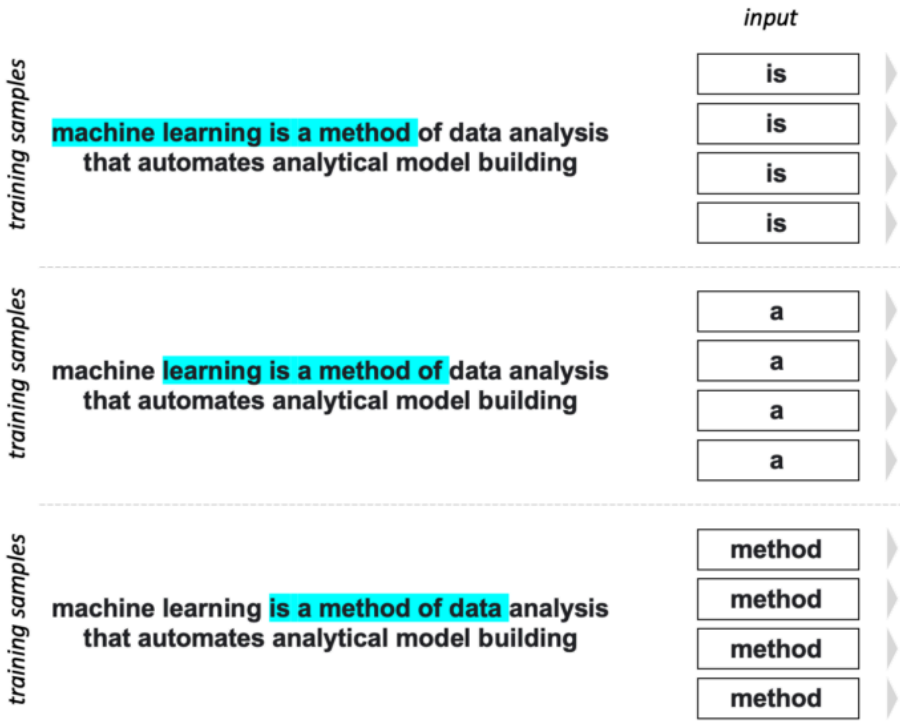


Image 3. Skip-Gram Model: High-level Overview. Image

What is happening in the black box?

The initial step would be to encode all words wi
integer (index) that identifies word position in th
"Vocabulary" is a term to describe a set of uniqu

This set may be all words in the text or just the most frequent ones. More on that in Section "Data Preparation".

Word2vec model is very simple and has only two layers:

- **Embedding layer**, which takes word ID and returns its 300-dimensional vector. Word2vec embeddings are 300-dimensional, as authors proved this number to be the best in terms of embedding quality and computational costs. about embedding layer as a simple lookup ta weights, or as a linear layer without bias and
- Then comes the **Linear (Dense) layer** with a We create a model for a multi-class classific the number of classes is equal to the numbe vocabulary.

The difference between CBOW and Skip-Gram m number of input words. CBOW model takes seve through the same Embedding layer, and then wo vectors are averaged before going into the Linea Gram model takes a single word instead. Detaile in the images below.

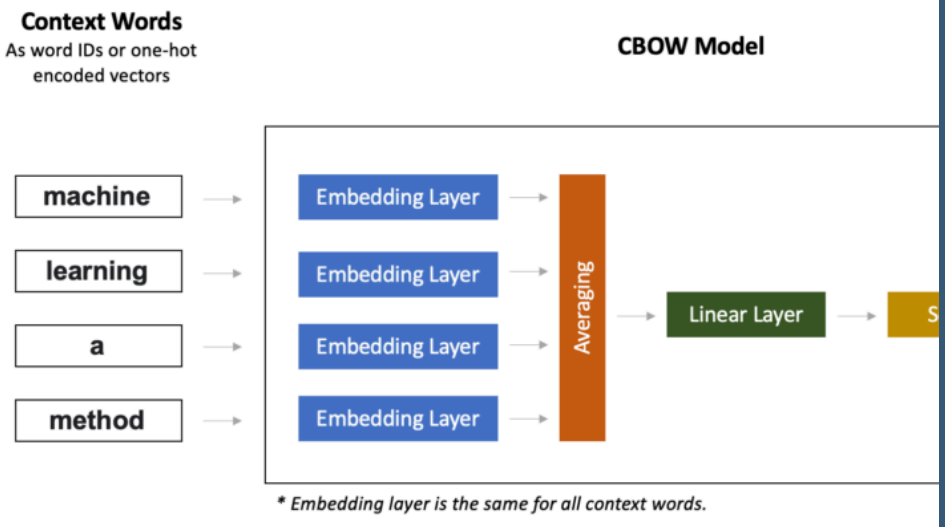


Image 4. CBOW Model: Architecture in Details. Image b

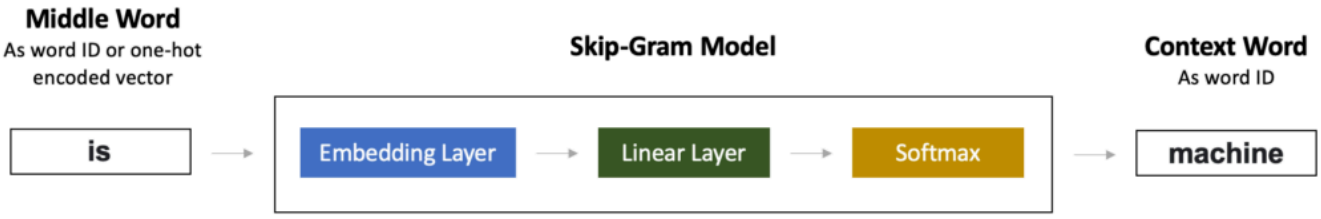


Image 5. Skip-Gram Model: Architecture in Details. Image by Author

Where are the word embeddings?

We train the models that are not going to be use
want to predict a word from its context or conte
Instead, we want to get word vectors. It turns ou
are weights of the Embedding layer. More details
Section "Retrieving Embeddings".

Data

Word2vec is an unsupervised algorithm, so we n
corpus. Originally, word2vec was trained on Goog
which contains 6B tokens.

I've experimented with smaller datasets availabl

- WikiText-2: 36k text lines and 2M tokens in t
words + punctuation)
- WikiText103: 1.8M lines and 100M tokens in t

When training word embedding for the commerc
choose the dataset carefully. For instance, if you
Machine Learning papers, train word2vec on scie
Machine Learning. If you'd like to classify fashion
of fashion news would be a better fit. That's bec
"model" means "approach" and "algorithm" in th
domain, but "person" and "woman" in the fashio

When reusing trained word embeddings, pay attention to the dataset they were trained on and whether this dataset is appropriate for your task.

Data Preparation

The main step in data preparation is to create a **vocabulary**. The vocabulary contains the words for which embed
Vocabulary may be the list of all the unique wor
corpus, but usually, it is not.

It is better to create vocabulary:

- Either by filtering out rare words, that occur
times in the corpus;
- Or by choosing the top N most frequent wor

Such filtering makes much sense because, with
vocabulary, the model is faster to train. On the c
probably do not want to use embedding for wor
only once within the text corpus, as these embe
good enough. To create good word embeddings t
see a word several times and in different contex

Each word in the vocabulary has its unique inde
vocabulary may be sorted alphabetically or base
frequencies, or may not – it should not affect th
Vocabulary is usually represented as a dictionary

```
vocab = {
    "a": 1,
    "analysis": 2,
    "analytical": 3,
```

```
    "automates": 4,
    "building": 5,
    "data": 6,
    ...
}
```

Punctuation marks and other special symbols may be also added to the vocabulary, and we train embeddings for them as well. You may lowercase all the words, or train separate embeddings for `apple` and `Apple`; in some cases, it may be useful.

Depending on what you want your vocabulary (and embeddings) to be like – preprocess the text corpus. Lowercase or not, remove punctuation or not, and

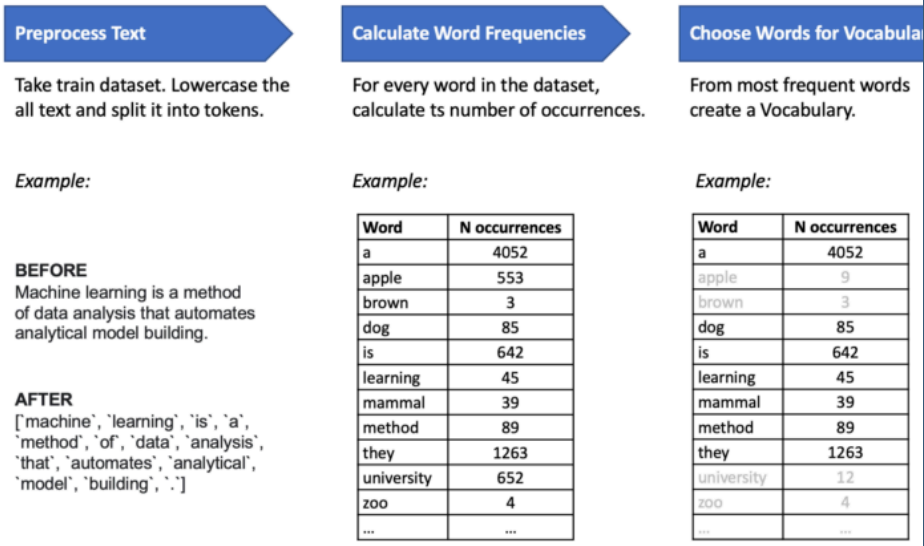


Image 6. How to create Vocabulary from a text corpus. Ima

For my model:

- I created vocabulary only from the words tha 50 times within a text.
- I used basic_english tokenizer from PyTorch text, splits it into tokens by whitespace, but into separate tokens.

So, before words go into the model, they are encoded as IDs. The ID corresponds to the word index in the vocabulary. Words that are not in the vocabulary (out-of-vocabulary words) are encoded with some number, for instance, 0.

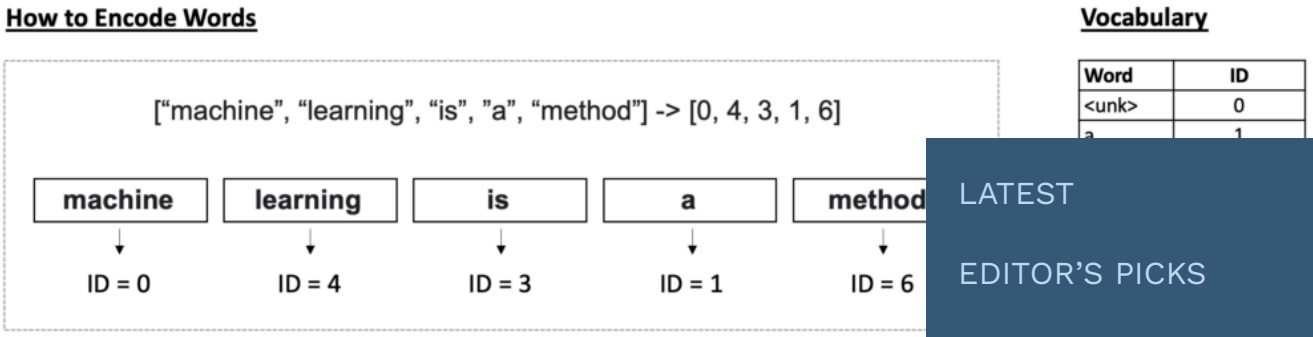


Image 7. How to Encode words with Vocabulary IDs. Imag

Text Processing with PyTorch

The full code for training word2vec is here. Let's important steps.

Models are created in PyTorch by subclassing fro described previously, both CBOW and Skip-Gram layers: Embedding and Linear.

Below is the model class for CBOW, and here is

```
import torch.nn as nn
EMBED_DIMENSION = 300
EMBED_MAX_NORM = 1

class CBOW_Model(nn.Module):
    def __init__(self, vocab_size: int):
        super(CBOW_Model, self).__init__()
        self.embeddings = nn.Embedding(
            num_embeddings=vocab_size,
```

```
            embedding_dim=EMBED_DIMENSION,
            max_norm=EMBED_MAX_NORM,
        )
        self.linear = nn.Linear(
            in_features=EMBED_DIMENSION,
            out_features=vocab_size,
        )
    def forward(self, inputs_):
        x = self.embeddings(inputs_)
        x = x.mean(axis=1)
        x = self.linear(x)
        return x
```

Pay attention, there is no Softmax activation in t
That's because PyTorch CrossEntropyLoss expec
raw, unnormalized scores. While in Keras you ca
the input to CrossEntropyLoss would be – raw v
probabilities.

Model input is word ID(s). Model output is an N-
where N is vocabulary size.

EMBED_MAX_NORM is the parameter to restrict
norms (to be 1, in our case). It works as a regula
and prevents weights in Embedding Layer grow u
EMBED_MAX_NORM is worth experimenting with
when restricting embedding vector norm, simila
and "father" have higher cosine similarity, compa
EMBED_MAX_NORM=None.

We create **vocabulary** from the dataset iterator
function build_vocab_from_iterator. WikiText-2 a
datasets have rare words replaced with token , v

a special symbol with ID=0 and all out-of-vocabulary words also encode with ID=0.

```
from torchtext.vocab import build_vocab_from_iterator MIN_WORD_FREQUENCY

def build_vocab(data_iter, tokenizer):
    vocab = build_vocab_from_iterator(
        map(tokenizer, data_iter),
        specials=["<unk>"],
        min_freq=MIN_WORD_FREQUENCY,
    )
    vocab.set_default_index(vocab["<unk>"])
    return vocab
```

Dataloader we create with collate_fn. This funct
logic of how to batch individual samples. When l
PyTorch WikiText-2 and WikiText103 datasets, ea
is a text paragraph.

For instance, in collate_fn for CBOW we "say":

1. Take each text paragraph.
    - Lowercase it, tokenize it, and encode it
    text_pipeline).
    - If the paragraph is too short – skip it. If
    it.
    - With the moving window of size 9 (4 his
    word, and 4 future words) loop through
    - All middle words merge into a list – they
    - All contexts (history and future words) n
    lists – they will be Xs.
2. Merge Xs from all paragraphs together – the

3. Merge Ys from all paragraphs together – they will be batch Ys.

Pay attention, the number of final batches (Xs and Ys) when we call collate_fn will be different from parameter batch_size specified in Dataloader, and will vary for different paragraphs.

Code for collate_fn for CBOW is below, for Skip-Gram – is here.

```
import torch CBOW_N_WORDS = 4
MAX_SEQUENCE_LENGTH = 256

def collate_cbow(batch, text_pipeline):
    batch_input, batch_output = [], []
    for text in batch:
        text_tokens_ids = text_pipeline(text)

        if len(text_tokens_ids) < CBOW_N_WORDS *
            continue

        if MAX_SEQUENCE_LENGTH:
            text_tokens_ids = text_tokens_ids[:MA

        for idx in range(len(text_tokens_ids) -
            token_id_sequence = text_tokens_ids[
            output = token_id_sequence.pop(CBOW_N
            input_ = token_id_sequence
            batch_input.append(input_)
            batch_output.append(output)

    batch_input = torch.tensor(batch_input, dtype
    batch_output = torch.tensor(batch_output, dty
    return batch_input, batch_output
```

And here is how to used collate_fn with PyTorch

```
from torch.utils.data
import DataLoader
from functools import partial

dataloader = DataLoader(
        data_iter,
        batch_size=batch_size,
        shuffle=True,
        collate_fn=partial(collate_cbow, text_pi
)
```

I've also created a class Trainer, that is used for
validation. It contains a typical PyTorch train and
for those who have experience with PyTorch, it w
straightforward.

If you want to understand the code better – I re
my repository and play with it.

Training Details

Word2vec is trained as a multi-class classificati
Cross-Entropy loss.

You choose batch size to fit into the memory. Ju
batch size is the number of dataset paragraphs,
processed into input-output pairs, and this num
larger.

The paper optimizer is AdaGrad, but I've used a
Adam.

I've skipped the paper part with Hierarchical Sof
plain Softmax. No Huffman tree was used either

vocabulary. Hierarchical Softmax and Huffman tree – are tricks for speeding up the training. But PyTorch has a lot of optimization under the hood, so training is already fast enough.

As recommended in the paper, I've started with a learning rate of 0.025 and decreased it linearly every epoch until it reaches 0 at the end of the last epoch. Here PyTorch LambdaLR scheduler helps a lot; and here is how I used it.

The authors trained the model for only 3 epochs
experiments (but on a very large dataset). I've ex
number smaller and larger and decided to stick

For WikiText-2 dataset:

- My vocabulary size was about 4k words (the
occurred in the text at least 50 times).
- Training took less than 20 minutes on GPU f
Skip-Gram models.

For WikiText103 dataset:

- My vocabulary size was about 50k words.
- The model trained overnight.

Retrieving Embeddings

The full procedure is described in this notebook.

Word embeddings are stored in the Embedding l
layer size is (vocab_size, 300), which means ther
embedding for all the words in the vocabulary.

When trained on the WikiText-2 dataset both CBOW and Skip-Gram models have weights in the Embedding layer of size (4099, 300), where each row is a word vector. Here is how to get Embedding layer weights:

```
embeddings = list(model.parameters())[0]
```

And here is how to get words in the same order as in the embedding matrix:

```
vocab.get_itos()
```

Before using embedding in your model, it's worth
they were trained properly. There are several opt

1. Cluster word embeddings and check if relate
separate clusters.
2. Visualize word embedding with t-SNE and cl
words lie close to each other.
3. Find the most similar words for a random w

Visualization with t-SN

You may use sklearn t-SNE and plotly to create a
visualization like the one below. Here numeric st
and they form 2 separate clusters.

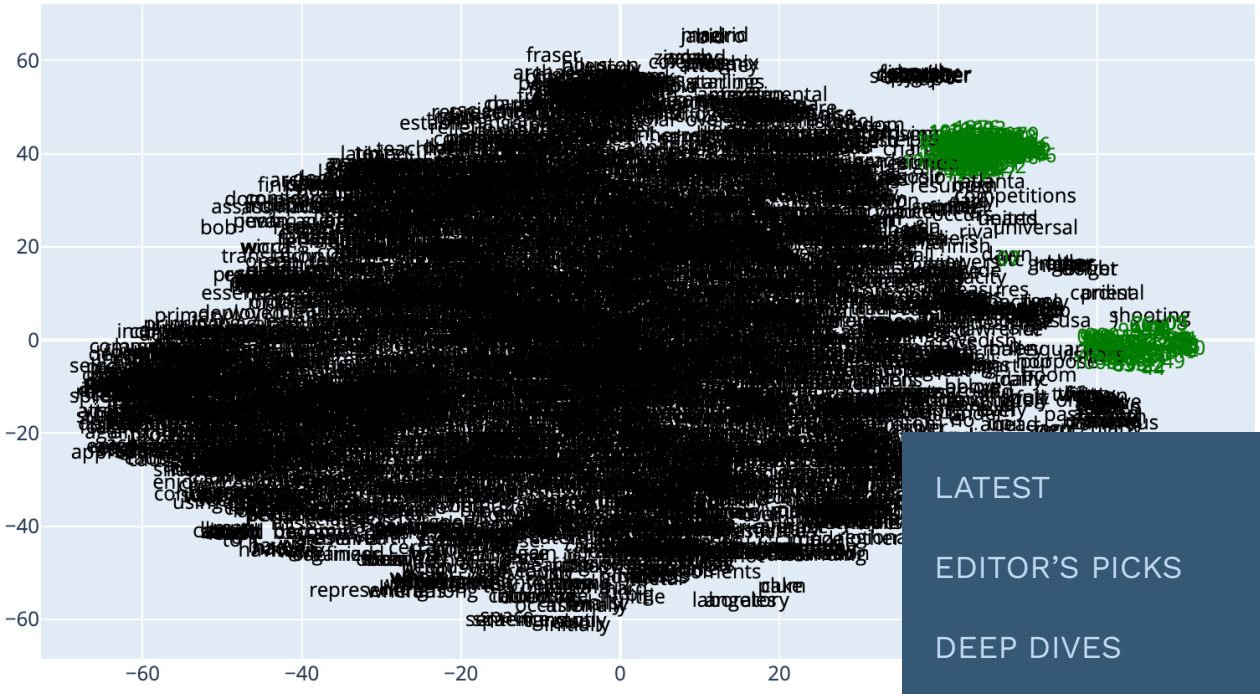
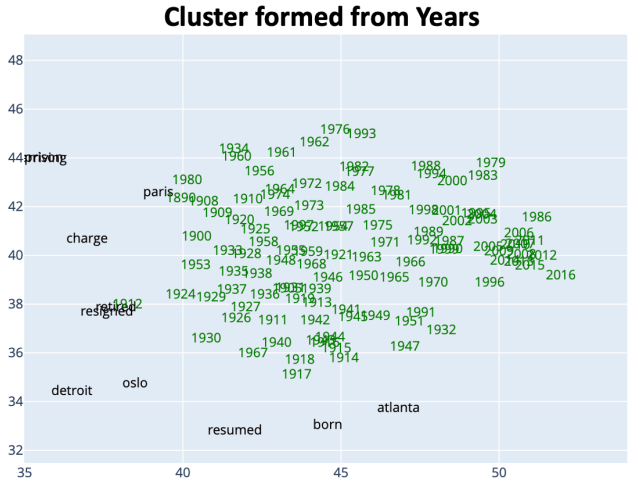


Image 8. Visualization of CBOW embeddings trained on WikiText-2 corpus. 2-com
plotted in green.

After zooming in, we may see that this division o
into 2 clusters makes much sense. The top left
from years, while the lower right – from plain nu



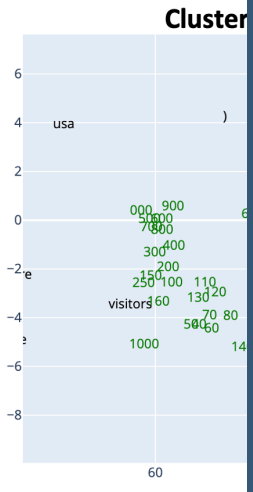


Image 9. Numeric clusters when zoomed in. The top left cluster is formed from ye
plain numbers.

You may explore this plotly visualization to find even more interesting relations. And the code for creating this visualization is here.

Similar Words

Word similarity is calculated as cosine similarity between word vectors. The higher the cosine similarity, the more similar words are assumed to be, obviously.

For instance, for the word "father" here are the r
in the vocabulary:

```
#CBOW model trained on WikiText-2
mother: 0.842
wife: 0.809
friend: 0.796
brother: 0.775
daughter: 0.773

#Skip-Gram model trained on WikiText-2
mother: 0.626
brother: 0.600
son: 0.579
wife: 0.563
daughter: 0.542
```

Code for finding similar words is here.

King – Man + Woman = Queen

According to the paper, a properly trained word2
equations "king – man + woman = ?" (answer: "q

big + small = ?" (answer: "smaller"), or "Paris – France + Germany = ?" (answer: "Berlin").

Equations are solved by performing mathematical operations on the word vectors: vector("king") – vector("man") + vector("woman"). And the final vector should be the closest to the vector("queen").

Unfortunately, I could not reproduce that part. My CBOW and Skip-Gram models trained on WikiText-2 and WikiText catch this kind of relation (code here).

The closest vectors to vector("king") – vector("m vector("woman") are:

```
#CBOW model trained on WikiText-2 dataset
king: 0.757
bishop: 0.536
lord: 0.529
reign: 0.519
pope: 0.501

#Skip-Gram model trained on WikiText-2 dataset
king: 0.690
reign: 0.469
son: 0.453
woman: 0.436
daughter: 0.435

#CBOW model trained on WikiText103 dataset
king: 0.652
woman: 0.494
queen: 0.354
daughter: 0.342
couple: 0.330
```

The closest vectors to vector("bigger") – vector("big") + vector("small") are:

```
#CBOW model trained on WikiText-2 dataset
small: 0.588
<unk>: 0.546
smaller: 0.396
architecture: 0.395
fields: 0.385

#Skip-Gram model trained on WikiText-2 dataset
small: 0.638
<unk>: 0.384
wood: 0.373
large: 0.342
chemical: 0.339

#CBOW model trained on WikiText103 dataset
bigger: 0.606
small: 0.526
smaller: 0.273
simple: 0.258
large: 0.258
```

Sometimes, correct words are within the top5, b
ones. I assume there are two possible reasons f

1. Some error within a code. To double-check I
with Gensim library and on the same datase
WikiText103. Gensim word embeddings are a
these equations.
2. So more probable reason is that the dataset
WikiText-2 contains 2M tokens and WikiText

while the Google News corpus used in the paper contains 6B tokens, which is 60 (!!!) times larger.

Dataset size really matters while training word embedding. And authors also mentioned that in the paper.

What's Next?

I hope this post helps you to build a foundation
can move on to more advanced algorithms. For
interesting to dig deep into the original paper an
from scratch. Recommended.

---

*Originally published at https://notrocketscience.b
29, 2021. If you'd like to read more tutorials like t
blog "Not Rocket Science" – Telegram and Twitte*

WRITTEN BY

Olga Chernytska

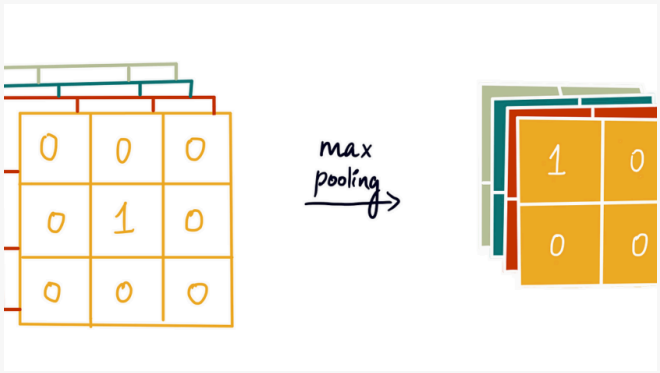
See all from Olga Chernytska

Topics:

Deep Dives | Deep Learning | Naturallanguageproc | Thoughts And Theory | Word2vec

Share this article:

Related Articles



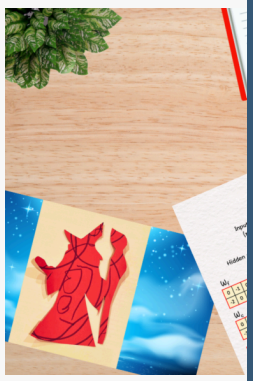
ARTIFICIAL INTELLIGENCE

Implementing Convolutional Neural Networks in TensorFlow

Step-by-step code guide to building a Convolutional Neural Network

Shreya Rao

August 20, 2024 • 6 min read



DEEP LEARNING

Deep Dive in by Hand ✍️

Explore the wis xLSTMs - a prob present-day LL

Srijanie Dey, PhD

July 9, 2024 • 13

DEEP LEARNING

Speeding Up the Vision Transformer with BatchNorm

How integrating Batch Normalization in an encoder-only Transformer architecture can lead to reduced training time...

Anindya Dey, PhD

August 6, 2024 • 28 min read

DATA SCIENCE

The Math Be Optimizers: and Applicat

This is a bit diff books say.

Peng Qian

August 17, 2024 • 9 min read

DATA SCIENCE

Latest picks: Time Series Forecasting with Deep Learning and Attention Mechanism

Your daily dose of data science

TDS Editors

November 4, 2020 • 1 min read

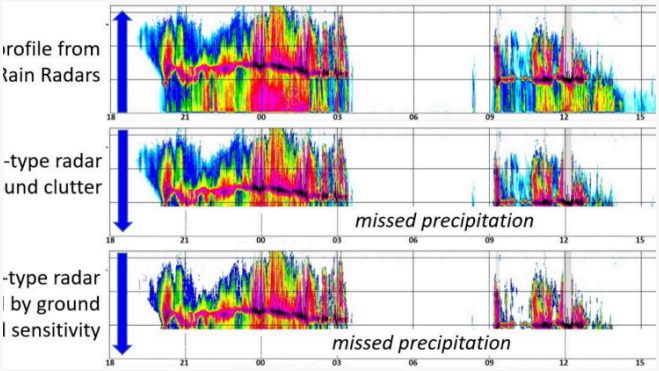
DATA SCIENCE

Stacked Ens… Advanced Pr… With H2O.ai

And how I place… largest machine… with them!

Sheila Teo

December 18, 202…



DEEP LEARNING

Beyond the Blind Zone

Inpainting radar gaps with deep learning

Fraser King

April 5, 2024 • 25 min read