word2vec

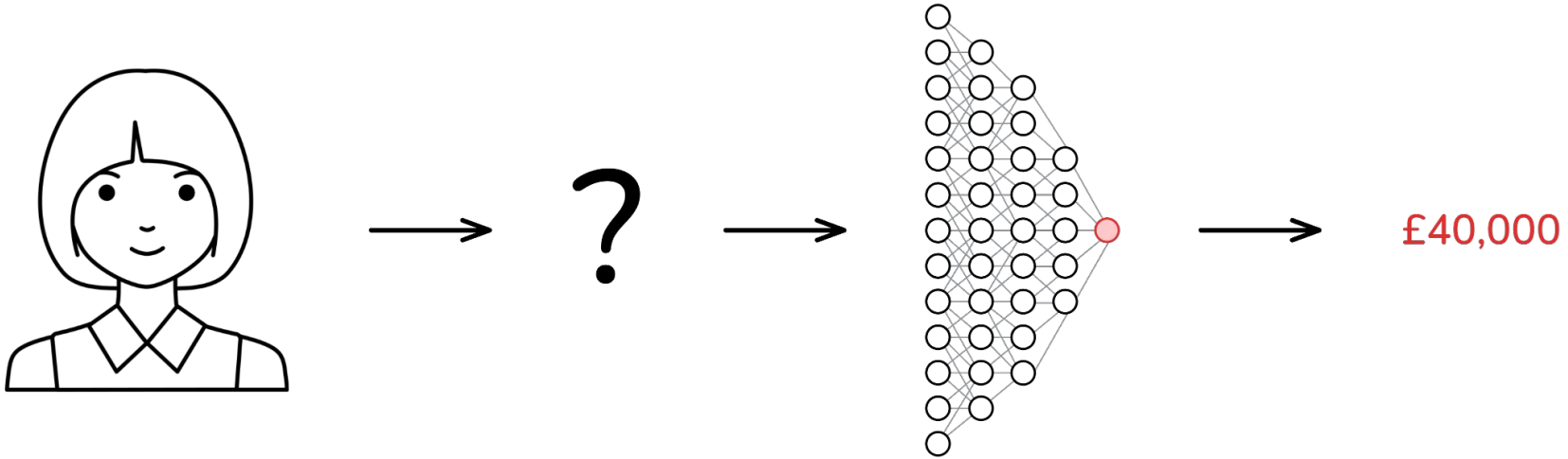
Word features

FOUNDERS AND CODERS

Machine Learning Institute

Income prediction model

Income prediction model

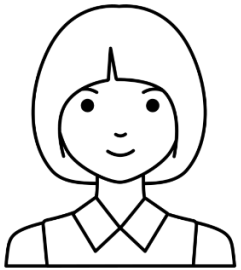


Convert the user's key characteristics into meaningful numbers the model can process

FOUNDERS AND CODERS

Meaningful numbers

Machine Learning Institute



Gender?	→	Yes, "0" is male, 1 is "female"
Age?	→	Yes, nice integer, between 0 and 100
Height?	→	Yep, use centimetres, between [20, 250]
Weight?	→	Yep, maybe just kilograms, [2, 150]
Education?	→	Hmm ...
City?	→	Oops ...
Job?	→	Oops ...

$$\begin{bmatrix} 1 \\ 28 \\ 170 \\ 62 \\ ? \\ X \\ X \end{bmatrix}$$

FOUNDERS AND CODERS

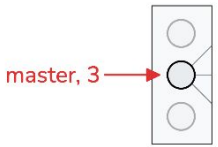
How to encode education?

Machine Learning Institute

Ordinal

0 - none
1 - high-school
2 - bachelor
3 - master
4 - PhD

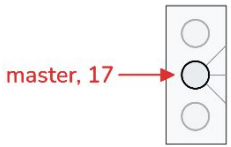
treat "PhD" as 4 times “none”



Years of Schooling

0 - none
12 - high-school
15 - bachelor
17 - master
21 - PhD

requires some domain knowledge



One-Hot

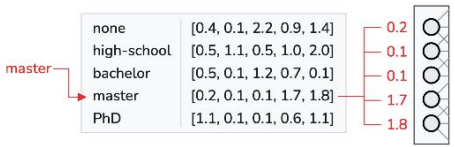
[1, 0, 0, 0, 0] - none
[0, 1, 0, 0, 0] - high-school
[0, 0, 1, 0, 0] - bachelor
[0, 0, 0, 1, 0] - master
[0, 0, 0, 0, 1] - PhD

no assumptions, use more neurons

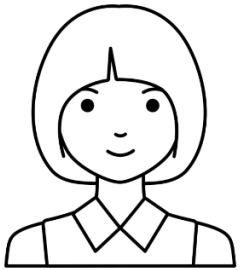
Learned Embeddings

[0.4, 0.1, 2.2, 0.9, 1.4] - none
[0.5, 1.1, 0.5, 1.0, 2.0] - high-school
[0.5, 0.1, 1.2, 0.7, 0.1] - bachelor
[0.2, 0.1, 0.1, 1.7, 1.8] - master
[1.1, 0.1, 0.1, 0.6, 1.1] - PhD

1 - come up with some random numbers i.e. initial features
2 - treat those features as trainable parameters
3 - more parameters, more data but best accuracy



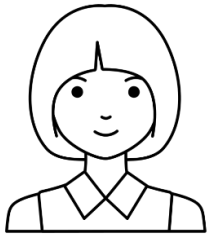
What about city?



Gender?	→	Yes, "0" is male, 1 is "female"
Age?	→	Yes, nice integer, between 0 and 100
Height?	→	Yep, use centimetres, between [20, 250]
Weight?	→	Yep, maybe just kilograms, [2, 150]
Education?	→	3, master
City?	→	Oops ...
Job?	→	Oops ...

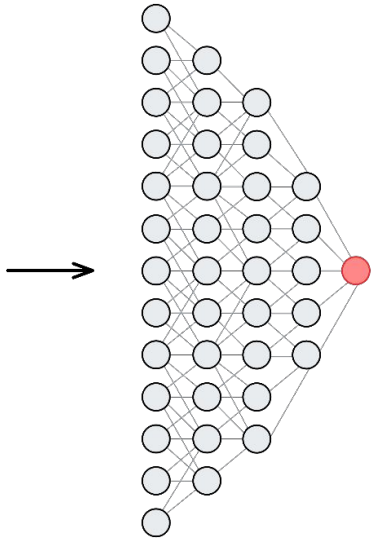
$$\begin{bmatrix} 1 \\ 28 \\ 170 \\ 62 \\ 3 \\ X \\ X \end{bmatrix}$$

Recursive Approach



Gender? → 1
Age? → 28
Height? → 170
Weight? → 62
Education? → 3
City?
- Population size? → 1M
- Average age? → 31
- Average income? → 22k
- Crime rate? → 4

[1, 28, 170, 62, 3, 1M, 31, 22k, 4]



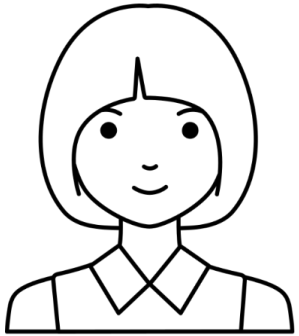
£40,000

FOUNDERS AND CODERS

Machine Learning Institute

One more feature...

One more feature…

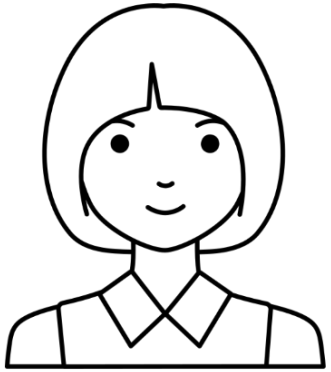


Free-text list of hobbies, interests and life story → Oops … → [X, X, X, X, X, X]

> Since cashing out of my fintech startup and upgrading to a Gulfstream G700 for family trips, I now channel my days into funding open-source AI research, sketching kinetic sculptures powered by custom Arduino boards, and sneaking away for dawn bouldering sessions before screening obscure Czech New-Wave films with my indie-cinema collective.

How can we produce meaningful numbers from free text?

One more feature…



Gender? → Yes, "0" is male, 1 is "female"

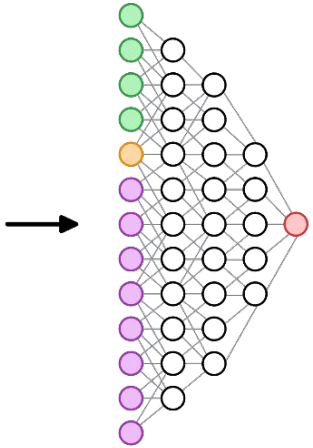
Age? → Yes, nice integer, between 0 and 100

Height? → Yep, use centimetres, between [20, 250]

Weight? → Yep, maybe just kilograms, [2, 150]

Education? → 3, master

Description → Since cashing out of my fintech startup...



£40,000

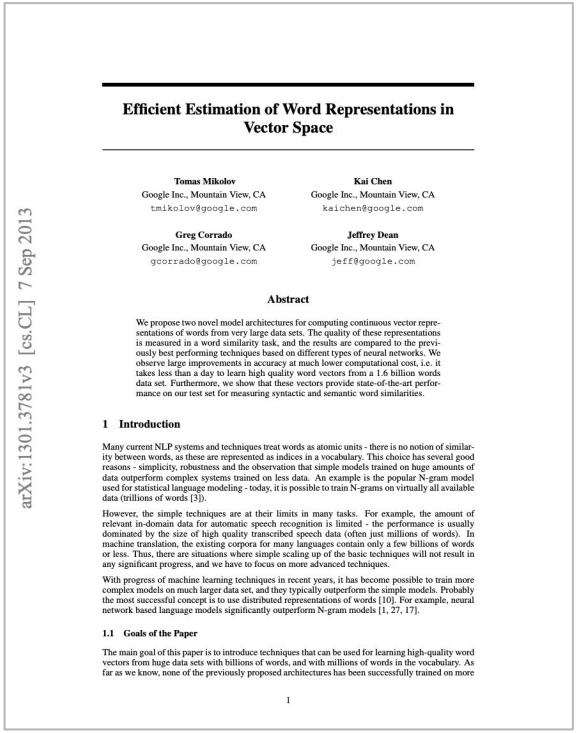
How can we produce meaningful numbers from free text?

Distribution hypothesis

What is the meaning of bardiwac?

- He handed her her glass of bardiwiac.
- Beef dishes are made to complement the bardiwiacs.
- Nigel staggered to his feet, face flushed from too much bardiwiac.
- I dined off bread and cheese and this excellent bardiwiac.

2013



Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

1 Introduction

Many current NLP systems and techniques treat words as atomic units - there is no notion of similarity between words, as these are represented as indices in a vocabulary. This choice has several good reasons - simplicity, robustness and the observation that simple models trained on huge amounts of data outperform complex systems trained on less data. An example is the popular N-gram model used for statistical language modeling - today, it is possible to train N-grams on virtually all available data (trillions of words [3]).

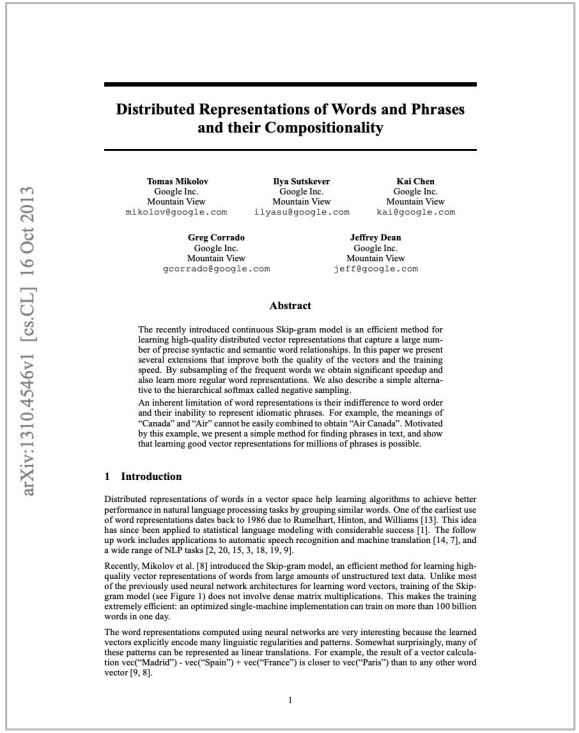
However, the simple techniques are at their limits in many tasks. For example, the amount of relevant in-domain data for automatic speech recognition is limited - the performance is usually dominated by the size of high quality transcribed speech data (often just millions of words). In machine translation, the existing corpora for many languages contain only a few billions of words or less. Thus, there are situations where simple scaling up of the basic techniques will not result in any significant progress, and we have to focus on more advanced techniques.

With progress of machine learning techniques in recent years, it has become possible to train more complex models on much larger data set, and they typically outperform the simple models. Probably the most successful concept is to use distributed representations of words [10]. For example, neural network based language models significantly outperform N-gram models [1, 27, 17].

1.1 Goals of the Paper

The main goal of this paper is to introduce techniques that can be used for learning high-quality word vectors from huge data sets with billions of words, and with millions of words in the vocabulary. As far as we know, none of the previously proposed architectures has been successfully trained on more

Mikolov et al. 2013 (January)



Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Abstract

The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. In this paper we present several extensions that improve both the quality of the vectors and the training speed. By subsampling of the frequent words we obtain significant speedup and also learn more regular word representations. We also describe a simple alternative to the hierarchical softmax called negative sampling.

An inherent limitation of word representations is their indifference to word order and their inability to represent idiomatic phrases. For example, the meanings of "Canada" and "Air" cannot be easily combined to obtain "Air Canada". Motivated by this example, we present a simple method for finding phrases in text, and show that learning good vector representations for millions of phrases is possible.

1 Introduction

Distributed representations of words in a vector space help learning algorithms to achieve better performance in natural language processing tasks by grouping similar words. One of the earliest use of word representations dates back to 1986 due to Rumelhart, Hinton, and Williams [13]. This idea has since been applied to statistical language modeling with considerable success [1]. The follow up work includes applications to automatic speech recognition and machine translation [14, 7], and a wide range of NLP tasks [2, 20, 15, 3, 18, 19, 9].

Recently, Mikolov et al. [8] introduced the Skip-gram model, an efficient method for learning high-quality vector representations of words from large amounts of unstructured text data. Unlike most of the previously used neural network architectures for learning word vectors, training of the Skip-gram model (see Figure 1) does not involve dense matrix multiplications. This makes the training extremely efficient: an optimized single-machine implementation can train on more than 100 billion words in one day.

The word representations computed using neural networks are very interesting because the learned vectors explicitly encode many linguistic regularities and patterns. Somewhat surprisingly, many of these patterns can be represented as linear translations. For example, the result of a vector calculation vec("Madrid") - vec("Spain") + vec("France") is closer to vec("Paris") than to any other word vector [9, 8].

Mikolov et al. 2013 (October)

word2vec - recipe

1

Treat the entire corpus as one long string. Slide a window of five tokens across it; for each window, mask the middle token and use the remaining four to predict the "?"

For dinner we served dark red bardiwac with steak and mushrooms.

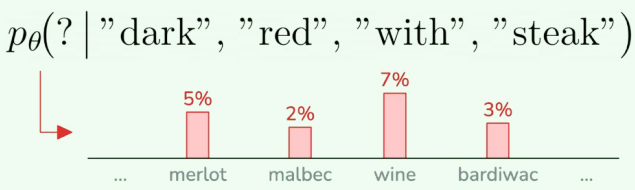
For dinner we served dark red bardiwac with steak and mushrooms.

dark red ? with steak

? = f(dark, red, with, steak)

2

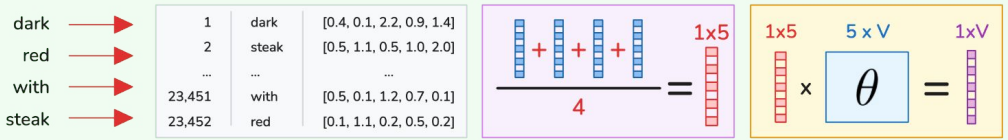
Because any given context can sensibly be completed by several different words, we don't force the model to output one "correct" token. Instead, we teach it to return a probability distribution over the whole vocabulary



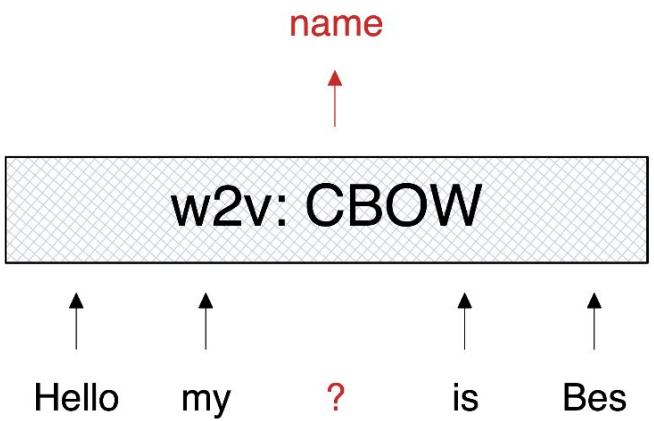
3

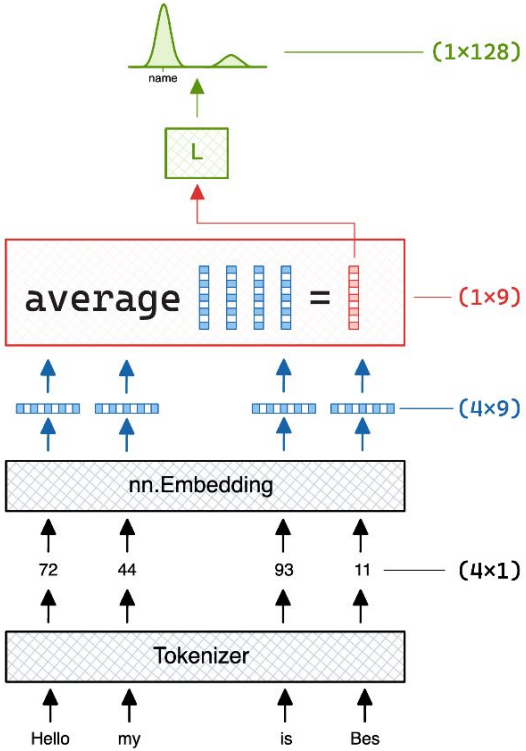
CBOW

Continuous Bag Of Words



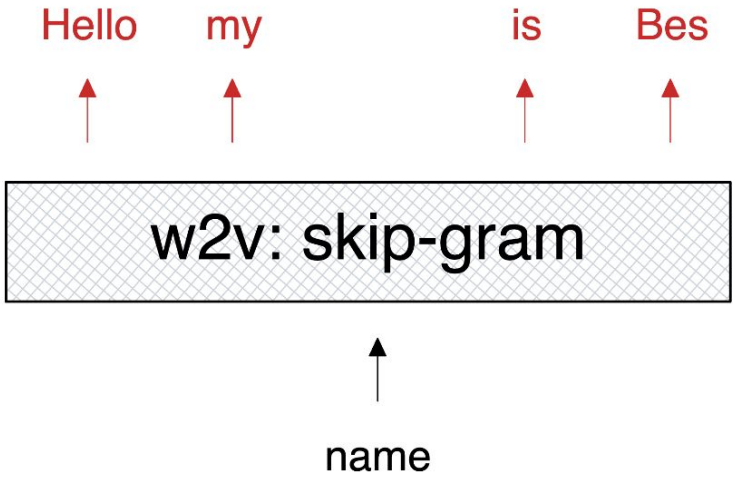
Continuous Bag of Words

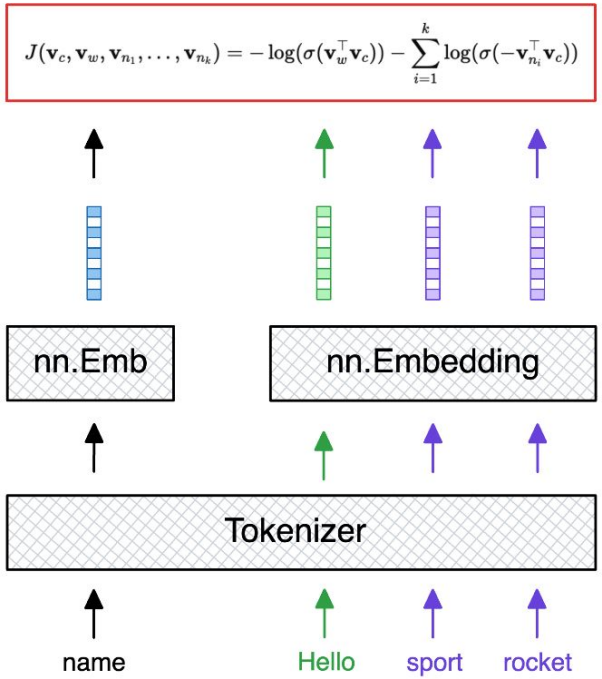




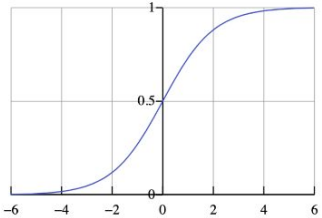

```python
#
#
#
import torch
import torch.nn.functional as F


#
#
#
vocab = {
  "Hello": 72,
  "my": 44,
  "name": 21,
  "is": 93,
  "Bes": 11
}

#
#
#
sentence = ["Hello", "my", "is", "Bes"]

#
#
#
class CBOW(torch.nn.Module):
  def __init__(self):
      super(CBOW, self).__init__()
      self.emb = torch.nn.Embedding(128, 9)
      self.linear = torch.nn.Linear(9, 128)

  def forward(self, inputs):
      embs = self.emb(inputs)
      embs = embs.mean(dim=1)
      out = self.linear(embs)
      probs = F.log_softmax(out, dim=1)
      return probs
```

Skip-Gram

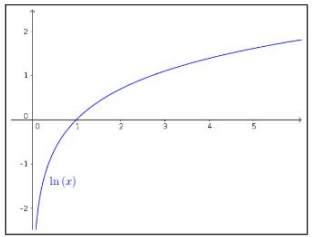




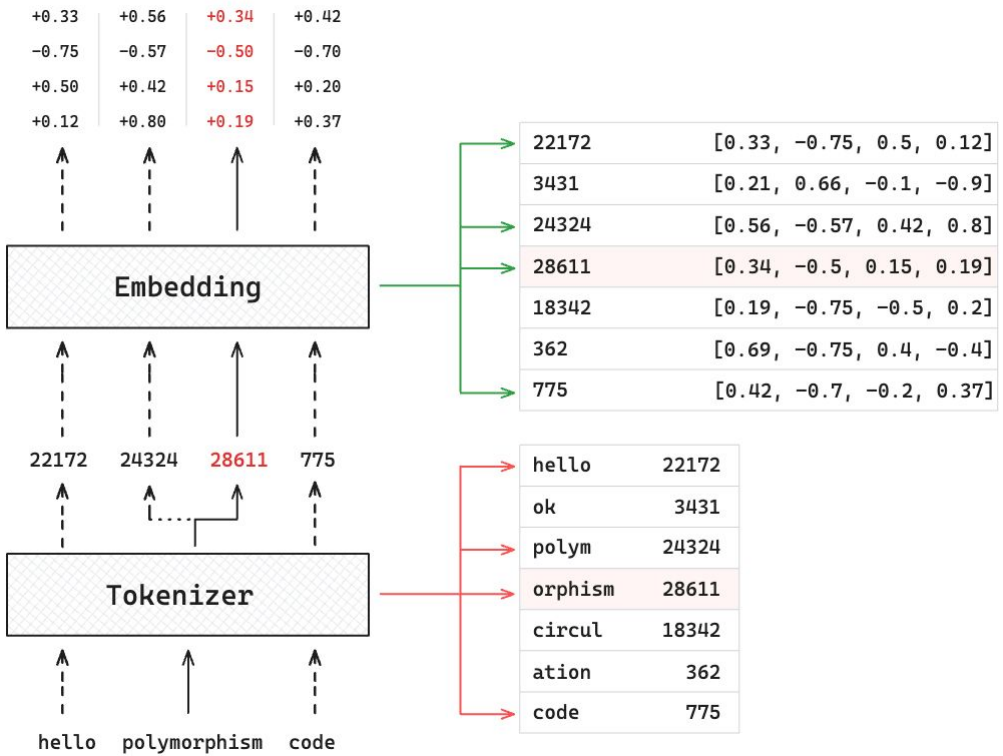
Sigmoid



Logarithm

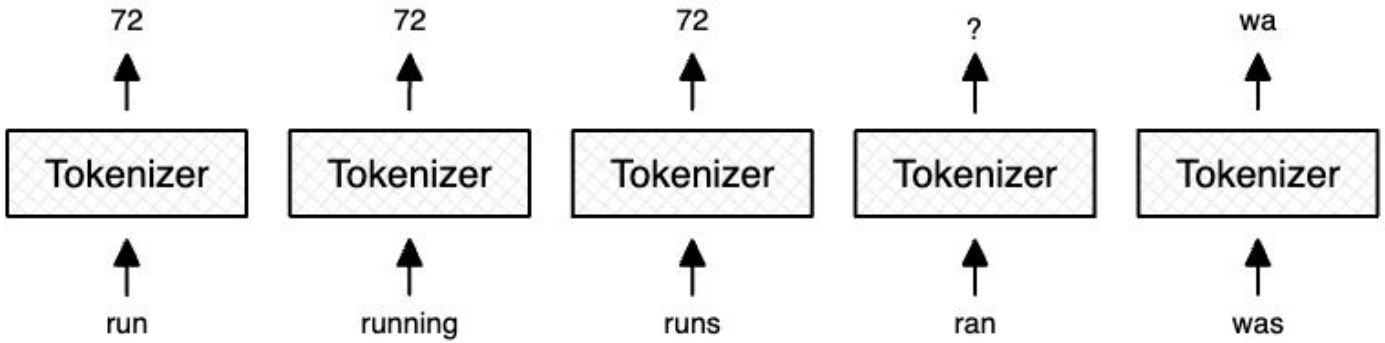


Tokenizer


+0.33 +0.56 +0.34 +0.42
−0.75 −0.57 −0.50 −0.70
+0.50 +0.42 +0.15 +0.20
+0.12 +0.80 +0.19 +0.37
Embedding
22172 24324 28611 775
Tokenizer
hello polymorphism code
22172 [0.33, −0.75, 0.5, 0.12]
3431 [0.21, 0.66, −0.1, −0.9]
24324 [0.56, −0.57, 0.42, 0.8]
28611 [0.34, −0.5, 0.15, 0.19]
18342 [0.19, −0.75, −0.5, 0.2]
362 [0.69, −0.75, 0.4, −0.4]
775 [0.42, −0.7, −0.2, 0.37]
hello 22172
ok 3431
polym 24324
orphism 28611
circul 18342
ation 362
code 775

Tokenizer

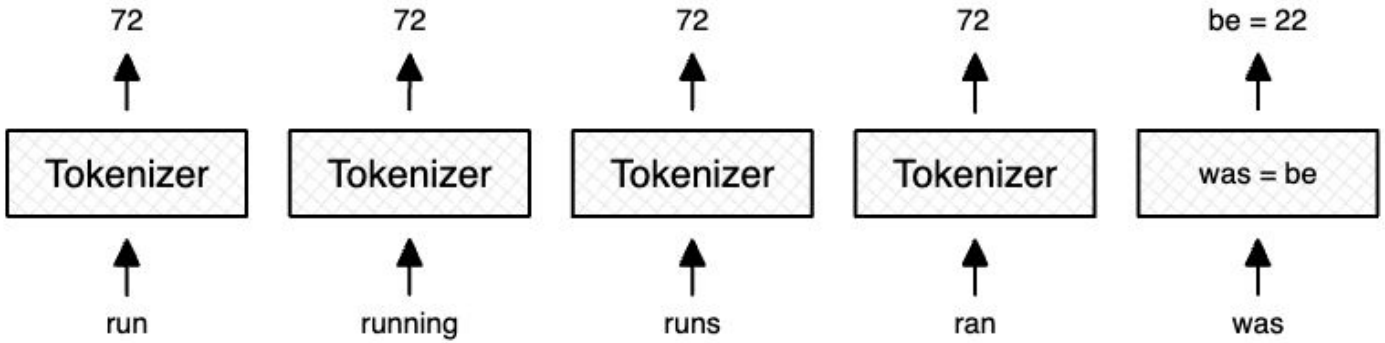
Stemming



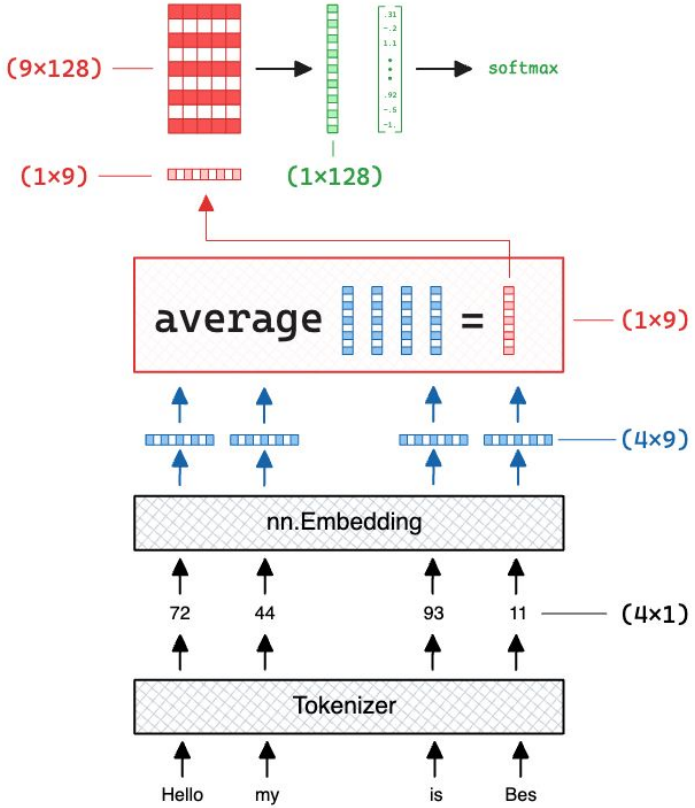

```
remove ["ing", "s"]
```

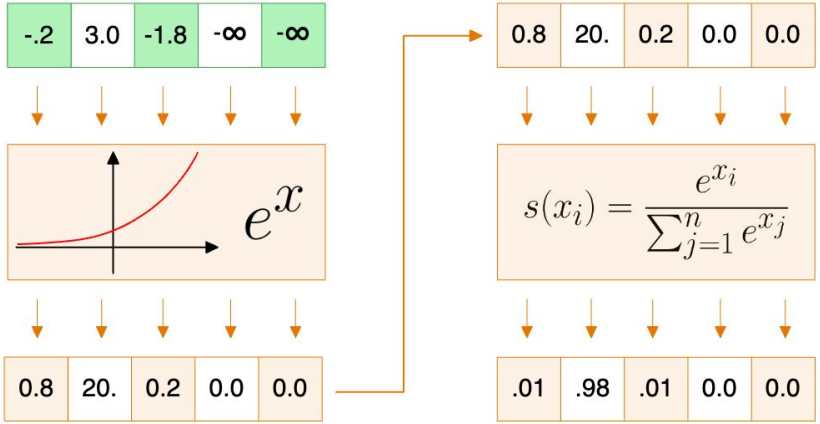
Tokenizer

Lemmatization



vocab mapping

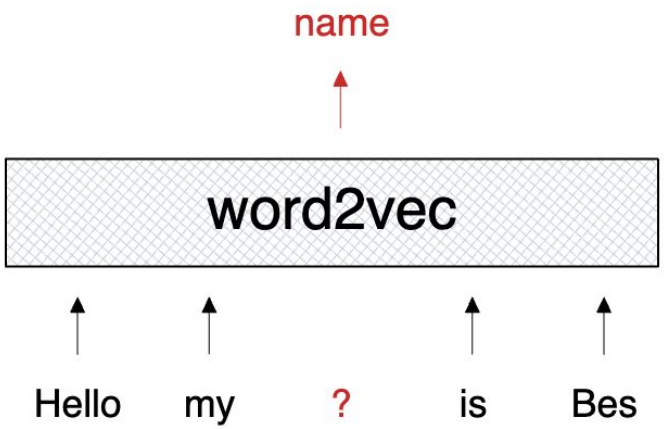

(9×128)
softmax
(1×9)
(1×128)
average
=
(1×9)
(4×9)
nn.Embedding
72
44
93
11
(4×1)
Tokenizer
Hello
my
is
Bes

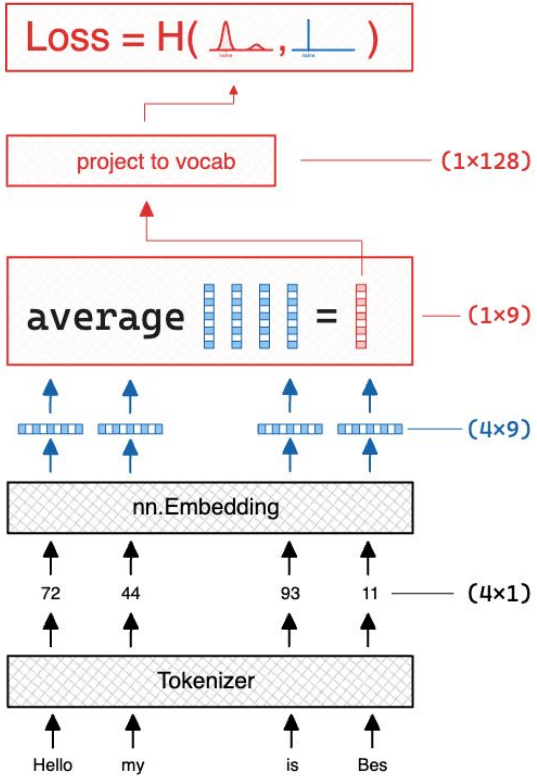

-.2
3.0
-1.8
-∞
-∞
e^x
0.8
20.
0.2
0.0
0.0
$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$
0.8
20.
0.2
0.0
0.0
.01
.98
.01
0.0
0.0

FOUNDERS AND CODERS

Overview

Machine Learning Institute



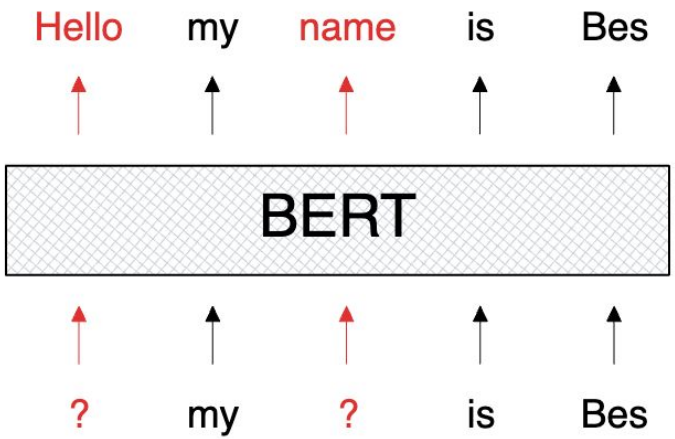


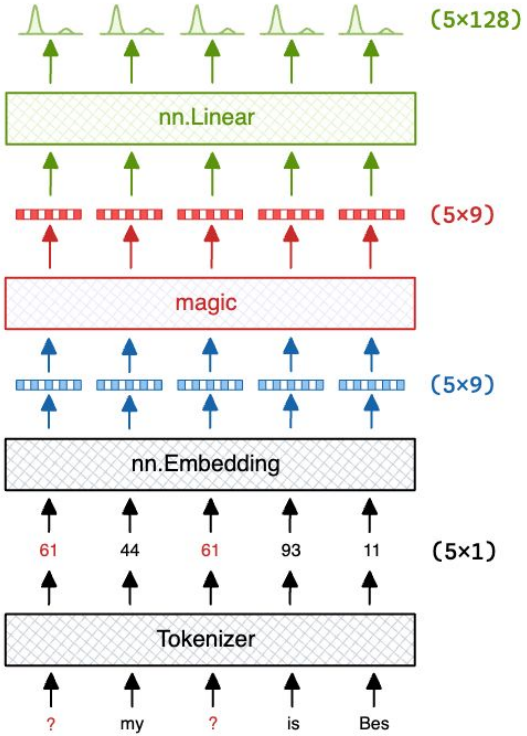

```python
#
#
#
import torch
import torch.nn.functional as F

#
#
#
vocab = {
  "Hello": 72,
  "my": 44,
  "name": 21,
  "is": 93,
  "Bes": 11
}

#
#
#
sentence = ["Hello", "my", "is", "Bes"]

#
#
#
class CBOW(torch.nn.Module):  ⇥ tab
  def __init__(self):
    super(CBOW, self).__init__()
    self.emb = torch.nn.Embedding(128, 9)
    self.linear = torch.nn.Linear(9, 128)

  def forward(self, inputs):
    embs = self.emb(inputs)
    embs = embs.mean(dim=1)
    out = self.linear(embs)
    probs = F.log_softmax(out, dim=1)
    return probs

```

FOUNDERS AND CODERS

BERT

Machine Learning Institute



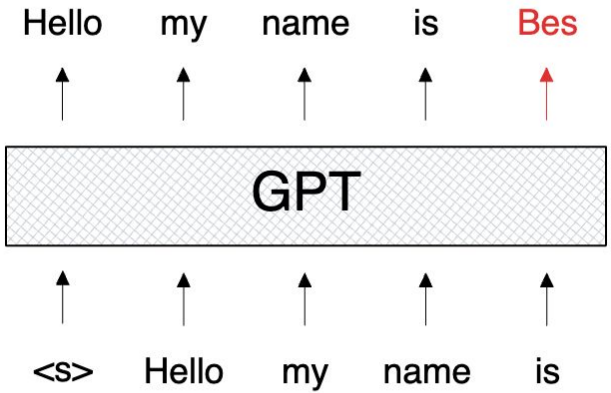


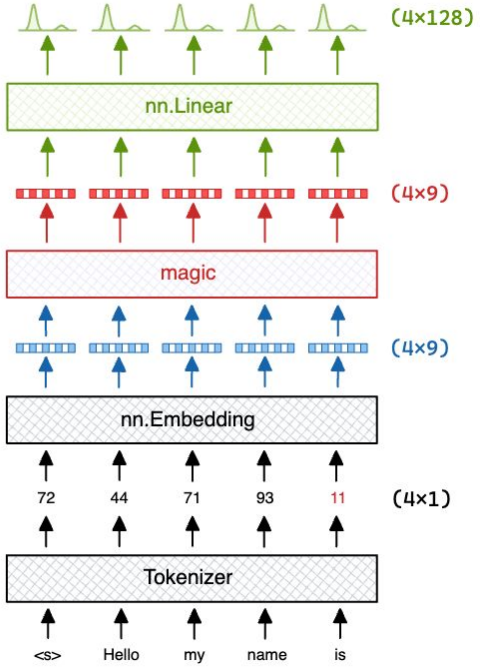

```python
#
#
import torch

#
#
class Block(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.W_Q = torch.nn.Linear(9, 9)
        self.W_K = torch.nn.Linear(9, 9)
        self.W_V = torch.nn.Linear(9, 9)
        self.ins = torch.nn.Linear(9, 9)
        self.rlu = torch.nn.ReLU()
        self.out = torch.nn.Linear(9, 9)

    def forward(self, x):
        Q = self.W_Q(x)
        K = self.W_K(x)
        V = self.W_V(x)
        x = Q @ K.T
        x = x @ V
        x = self.out(x)
        x = self.rlu(x)
        x = self.ins(x)
        return x

#
#
class BERT(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.embed = torch.nn.Embedding(53, 9)
        block_layers = [Block() for _ in range(5)]
        self.blocks = torch.nn.ModuleList(block_layers)
        self.linear = torch.nn.Linear(9, 53)

    def forward(self, x):
        x = self.embed(x)
        x = x.squeeze(0)
        for block in self.blocks: x = block(x)
        x = self.linear(x)
        return x

```

FOUNDERS AND CODERS

Overview

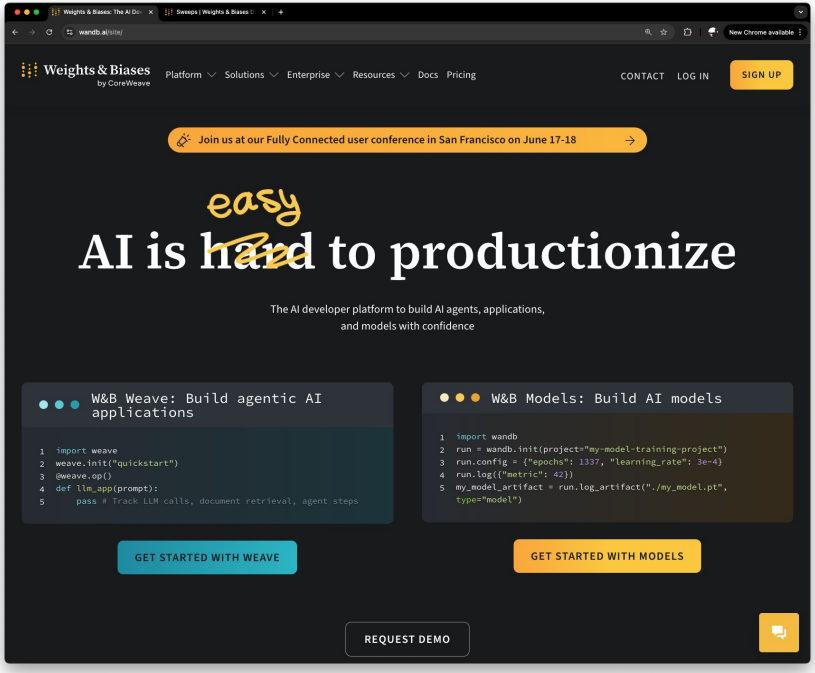
Machine Learning Institute

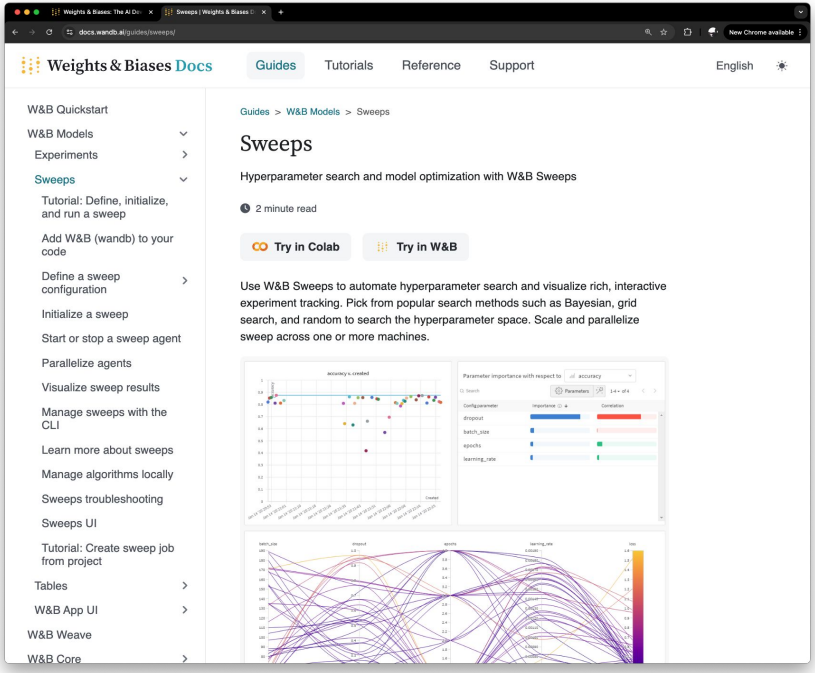





```python
import torch
import torch.nn.functional as F


vocab = {
  "?": 61,
  "Hello": 72,
  "my": 44,
  "name": 21,
  "is": 93,
  "Bes": 11
}


sentence = ["?", "my", "?", "is", "Bes"]


class Magic(torch.nn.Module):
  def __init__(self):
    super(Magic, self).__init__()
    ...
  def forward(self, inputs):
    ...


class BERT(torch.nn.Module):
  def __init__(self):
    super(BERT, self).__init__()
    self.emb = torch.nn.Embedding(128, 9)
    self.magics = torch.nn.ModuleList([Magic() for _ in range(3)])
    self.linear = torch.nn.Linear(9, 128)

  def forward(self, inputs):
    embs = self.emb(inputs)
    for magic in self.magics: embs = magic(embs)
    probs = F.log_softmax(embs, dim=1)
    return probs
```

Weights & Biases





FOUNDERS AND CODERS

Machine Learning Institute

Good luck!