

Dense representation for categorical variables

One drawback that we flagged with One Hot Encoding of a categorical variable

- Tokens that seem to be related in the input domain ("dog", "dogs")
- Become *unrelated* when One Hot Encoded

Because each value is a long vector

- With a single non-zero element
- That is *different* across the two values
- The OHE vectors are *orthogonal*

This means that there is no useful measure of the distance between two tokens

To illustrate the "lack of distance" issue, let rep be a mapping from tokens to their One Hot Encodings.

Using dot product (cosine similarity) as a measure of similarity to the token "dog"

word	$\text{rep}(\text{word})$	Similarity
dog	[1,0,0,0]	$\text{rep}(\text{word}) \cdot \text{rep}(\text{dog}) = 1$
dogs	[0,1,0,0]	$\text{rep}(\text{word}) \cdot \text{rep}(\text{dog}) = 0$
cat	[0,0,1,0]	$\text{rep}(\text{word}) \cdot \text{rep}(\text{dog}) = 0$
apple	[0,0,0,1]	$\text{rep}(\text{word}) \cdot \text{rep}(\text{dog}) = 0$

All words other than "dog" are equidistant from "dog".

Intuitively, we observe similarity between "dog" and other words

- ("dog", "dogs"): Same root, different Singular/Plural form
- ("dog", "cat"); Same concept: pet

and complete lack of similarity between "dog" and "apple".

Yet all have the same distance measure from "dog": 0.

We can consider an alternate encoding to OHE.

Suppose each dimension of the encoded vector

- Measured the intensity of the token against some concept
 - Singular/Plural
 - Domestic Animal
 - Edible

This type of representation is called *continuous*

- As the strength is a continuous value
- Compared to the *discrete* encoding of OHE as binary 0/1

It is also called a *dense* representation

- Multiple non-zero elements in the vector
- Compared to the single non-zero element in the OHE vector

In a continuous, dense representation two values expressing similar concepts will be "closer" than two values that do not share concepts

- "Cats", "Dogs", "Apples"
 - Share the concept "Plural"
- "Cat", "Dog"
 - Share the concept "Domestic animal"
- "good", "bad"
 - Share the concept "Opposite"

Doing math with words

Let's explore the implication and power of dense vector representation of words.

If each element of the vector

- Expresses a concept
- And the number of concepts is small compared to $||\mathbf{V}||$
- And the concepts are fairly independent
- Then we have found an alternate basis (compared to the $||\mathbf{V}||$ basis vectors of OHE) of smaller dimension
- For representing words

This concept is sometimes called *word embeddings*

Let \mathbf{v}_w denote the dense representation of token w :

w	\mathbf{v}_w
cat	[.7, .5, .01]
cats	[.7, .5, .95]
dog	[.7, .2, .01]
dogs	[.7, .2, .95]
apple	[.1, .4, .01]
apples	[.1, .4, .95]

Notice that "dogs" and "apples"

- Are similar along one dimension (the last, perhaps encoding "Is Plural")
- Are dissimilar along one dimension (the first, perhaps encoding "Is Pet")

Also notice that "dog" and "cat"

- Are similar along the first dimension (reinforcing the notion that this dimension may be "Is Pet")

Taking this a step further: we can perform element-wise math on dense vector representations:

$$\mathbf{v}_{\text{cats}} - \mathbf{v}_{\text{cat}} \approx \mathbf{v}_{\text{dogs}} - \mathbf{v}_{\text{dog}} \approx \mathbf{v}_{\text{apples}} - \mathbf{v}_{\text{apple}}$$

because

- "cats" and "cat" are similar in all concepts *except* "Plural".
- As are "dogs" and "dog"
- As are "apples" and "apple"

If that's the case, we can approximate the vector that expresses the "pure" concept "Is Plural"

- Without expressing any other concept

as $(\mathbf{v}_{\text{cats}} - \mathbf{v}_{\text{cat}})$

Then we can construct the Plural form of "apple"

- By adding the pure vector for Plural to the vector for "apple"

$$\mathbf{v}_{\text{apples}} \approx \mathbf{v}_{\text{apple}} + (\mathbf{v}_{\text{cats}} - \mathbf{v}_{\text{cat}})$$

we can create the Plural form of "apple"

Word analogies

The implications of doing math on words is even more powerful.

Consider solving the analogy problem

king:man :: ?:woman

That is: what is the female analog of "king" ?

Suppose the concepts ("dimensions") of the dense representation were

- Gender (man or woman)
- Regal (Royal or commoner)

Then

$$\mathbf{v}_{\text{king}} - \mathbf{v}_{\text{man}} + \mathbf{v}_{\text{woman}} \approx \mathbf{v}_{\text{queen}}$$

because

\mathbf{v}_{king}	$=$	$(\text{Man}, \text{Royal})$	vector representation
$\mathbf{v}_{\text{king}} - (\text{Man}, 0)$	$=$	$(0, \text{Royal})$	subtract vector for "Man"
$\mathbf{v}_{\text{king}} - (\text{Man}, 0) + (\text{Woman}, 0)$	$=$	$(\text{Woman}, \text{Royal})$	add vector for "pure" woman
	$=$	Queen	the word having connotation of "pure" woman

We can use math on dense vectors to compute analogies!

Let's formalize the "math" of word vectors

For tokens w, w' with dense vectors $\mathbf{v}_w, \mathbf{v}_{w'}$

- Define a metric $d(\mathbf{v}_w, \mathbf{v}_{w'})$ of the distance between the words
- For example:

$$d(\mathbf{v}_w, \mathbf{v}_{w'}) = 1 - \text{cosine similarity}(\mathbf{v}_w, \mathbf{v}_{w'})$$

Define the set of tokens $N_{n',d}(w)$ in vocabulary \mathbf{V}

- That are among the n' "closest" to a token w
- According to distance metric d

$$\begin{aligned}\mathbf{wv}_{n',d}(w) &= \{ \mathbf{v}_{w'} \mid \text{rank}_V(d(\mathbf{v}_w, \mathbf{v}_{w'})) \leq n' \} \quad \text{the dense vectors of the } n' \text{ to } w \\ N_{n',d}(w) &= \{ w' \mid \mathbf{v}_{w'} \in \mathbf{wv}_{n',d}(w) \}\end{aligned}$$

This is the "neighborhood" of token w as defined by the distance metric.

Token w' is defined to be *approximately equal* to token w

- Denoted as $w \approx_{n',d} w'$
- If w' is in the neighborhood of w

$$w \approx_{n',d} w' \text{ if } \mathbf{w}' \in N_{n',d}(w)$$

Thus, the analogy

$$a:b :: c:d$$

implies

$$\mathbf{v}_a - \mathbf{v}_b \approx_{n',d} \mathbf{v}_c - \mathbf{v}_d$$

So to solve the word analogy for c :

$$\mathbf{v}_c \approx_{n',d} \mathbf{v}_a - \mathbf{v}_b + \mathbf{v}_d$$

GloVe: Pre-trained embeddings

Fortunately, you don't have to create your own word-embeddings from scratch.

There are a number of pre-computed embeddings freely available.

GloVe is a family of word embeddings that have been trained on large corpora

- GloVe6b
 - Trained on 6 Billion tokens
 - 400K words
 - Corpus: Wikipedia (2014) + GigaWord5 (version 5, news wires 1994-2010)
 - Many different dense vector lengths to choose from
 - 50, 100, 200, 300

We will illustrate the power of word embeddings using GloVe6b vectors of length 100.

king- man + woman	$\approx_{n',d}$	queen
man - boy + girl	$\approx_{n',d}$	woman
Paris - France + Germany	$\approx_{n',d}$	Berlin
Einstein - science + art	$\approx_{n',d}$	Picasso

You can see that the dense vectors seem to encode "concepts", that we can manipulate mathematically.

You may discover some unintended bias

doctor - man + woman	$\approx_{n',d}$	nurse
mechanic - man + woman	$\approx_{n',d}$	teacher

Domain specific embeddings

Do we speak Wikipedia English in this room ?

Here are the neighborhoods of some financial terms, according to GloVe:

$N(\text{bull})$ = [cow, elephant, dog, wolf, pit, bear, rider, lion, horse]

$N(\text{short})$ = [rather, instead, making, time, though, well, longer, shorter, l

$N(\text{strike})$ = [workers, struck, action, blow, striking, protest, stoppage, wal

$N(\text{FX})$ = [showtime, cnbc, ff, nickelodeon, hbo, wb, cw, vh1]

It may be desirable to create word embeddings on a narrow (domain specific) corpus.

This is not difficult provided you have enough data.

Obtaining Dense Vectors: Transfer Learning

How do we obtain Dense Vector representations that seem to have these wonderful properties ?

- Through Machine Learning !
- As a by-product of solving a specific Source task
- Once we have the embeddings, we can re-use them in many other Target tasks.

This is exactly what we called Transfer Learning

- We train a Source Task
- The layers and associated weights learned in training the Source Task
- Are re-used for a different Target Task

The layer and associated weights that implement the dense vector encoding are re-used

- Called an *Embedding Layer*

We will show code that trains an Embedding Layer shortly.

Word prediction problems: high-level

The Source Task we will use to create word embeddings is from a class of *Word Prediction* tasks

- Given a set of tokens (the "context")
- Predict a related token

For example

- Given prefix $\mathbf{w}_{(1)} \dots \mathbf{w}_{(t-1)}$ of a sequence of tokens \mathbf{w}
- Predict the next token $\mathbf{w}_{(t)}$.

"Machine Learning is ⟨???

Or a similar problem

- Predict token $\mathbf{w}_{(t)}$
- From surrounding tokens
 $\mathbf{w}_{(t-o)}, \dots, \mathbf{w}_{(t-1)}, \langle ??? \rangle, \mathbf{w}_{(t+1)} \dots, \mathbf{w}_{(t+o)}$

"Machine $\langle ??? \rangle$ is easy"

The inspiration behind using a Word Prediction task to learn embeddings

- Is that meaning of a word can be inferred by context
- "You are known by the company that you keep"

For example

- "I ate an apple"
- "I ate a blueberry"
- "I ate a pie"

"apple", "blueberry", "pie" concept: things that you eat

The Word Prediction task is thus a form of Classification.

We need a large number of training examples as this is a Supervised Learning problem.

One reason that Word Prediction is used is that it is fairly easy to obtain training examples

- From any source of raw text
- Just reformat
- That is: the target/label for an example is just an adjacent token

Since targets can be derived from examples, this is sometimes called *Semi-Supervised Learning*

Let \mathbf{w} be the sequence of $n_{\mathbf{w}}$ words

A word prediction is a mapping

- from input \mathbf{w}
- to a probability distribution $\hat{\mathbf{y}}$ over all words in vocabulary \mathbf{V}
 - $\hat{\mathbf{y}}_j = p(V_j)$
 - That is: it assigns a probability to each word in the vocabulary

Here are some simple word prediction problems:

predict next word from context $p(\mathbf{w}_{(t)} \mid \mathbf{w}_{(t-o)} \dots, \mathbf{w}_{(t-1)})$

predict a surrounding word $p(\mathbf{w}_{(t')} \mid \mathbf{w}_{(t)})$

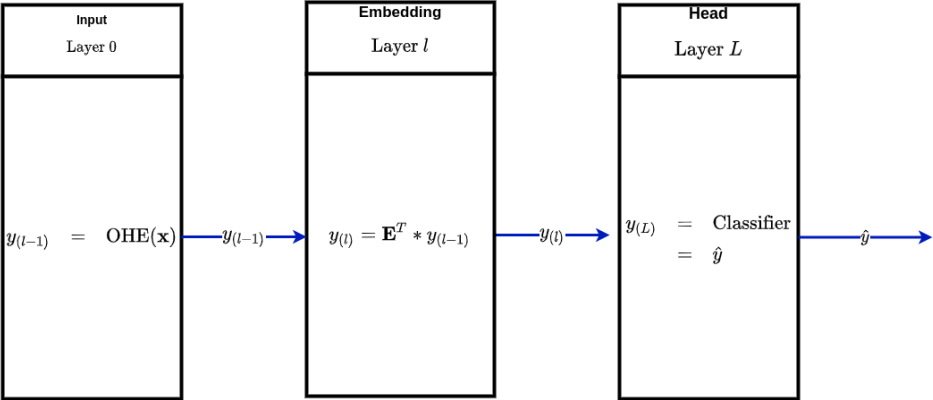
$$t' = \{t - o, \dots, t + o\} - \{t\}$$

predict center word from context $p(\mathbf{w}_{(t)} \mid [\mathbf{w}_{(t-o)} \dots \mathbf{w}_{(t-1)} \mathbf{w}_{(t+1)} \dots \mathbf{w}_{(t+o)}])$

Here is the Neural Network we construct for the Source Task that will learn embeddings

- Ignoring for the moment the issue of converting variable length sequences to a fixed length

Embedding Layer



$$\text{OHE}(\mathbf{x}) : (|\mathbf{V}| \times 1)$$

$$\mathbf{E} = \mathbf{W}_{(l)}$$
$$\mathbf{W}_{(l)} : (|\mathbf{V}| \times n_e)$$

$$\mathbf{W}_{(L)} : (n_e \times |\mathbf{V}|)$$

Layers:

- One Hot Encoded token
- Embedding: converts sparse encoding to dense encoding
- Classifier: operating on dense encodings

The only "new" layer type is the Embedding layer

- This is nothing more than Matrix Multiplication
- The mapping can be implemented as an $(|\mathbf{V}| \times n_e)$ matrix \mathbf{E}
- Where n_e is the length chosen for the dense vector

That is because

- The OHE vector for the j^{th} word \mathbf{V}_j in vocabulary \mathbf{V}
- Is the $(|\mathbf{V}| \times 1)$ vector of all 0's except at index j
 $V^{(j)} = 1$
- $\mathbf{E}^T * \text{OHE}(\mathbf{V}_j)$
- Selects row j of \mathbf{E} , which is the $(n_e \times 1)$ *dense vector* encoding of \mathbf{V}_j

Matrix **E** are *weights to be learned* by training

- Along with the weights of the Classifier layer

In other words

- We train the Neural Network
- To create an embedding
- That makes it easy for a Classifier
- To solve the Source Task

Conclusion

Categorical variables (such as tokens/words) are easily represented as One Hot Encoded values.

This is perfectly adequate when there is no relationship between tokens.

Word embeddings/Dense representations create a representation

- Not just of a token is isolation
- But a token with multiple dimensions of meaning
- Which enable inter-token relationships

We showed how to create dense representation of words as a by-product of solving a Source Task.

The Source Task we used was Word Prediction, but other tasks may work as well.

The embeddings learned for the Source Task may be useful in other tasks

- This is Transfer Learning in the real world

```
In [ ]: print("Done")
```