# Natural Language Processing

The datasets for Machine Learning have historically been mainly numeric.

But non-numeric data such as Image and Text is an abundant and potentially rich source of insight.

We have illustrated many of the concepts in this course with Image data.

We will briefly dive into the world of text.

*Natural Language Processing* is the set of tools/techniques that facilitate using text as raw material.

## The world of text

- SEC filing
- analyst reports
- news articles
- tweets

We will approach text mainly from a Deep Learning perspective

- lots of data
- minimal pre-processing
- "feature engineering" by the Neural Network

That is not to discount more "classical" methods for NLP

- Part of speech
- Stemming
- Lemmatization
- n-grams

All of these are potentially useful as pre-processing steps for Deep Learning.

However, if our data sets are big enough, it may be counter-productive to preprocess.

## Issues with text

There are several big issues to tackle regarding text data

- Words are categorical variables
- Token sequences (sentences/paragraphs) are variable length
- Token sequences: order matters

We are using the term "token" rather than word

- tokens may include punctuation, special characters
- tokens may be characters rather than entire words

## Notation

- $\mathbf{w}$ is a sequence of $n_{\mathbf{w}}$ tokens $\mathbf{w}_{(1)}, \dots, \mathbf{w}_{(n_{\mathbf{w}})}$
- each token is an element of vocabulary $\mathbf{V} : \mathbf{w}_{(t)} \in \mathbf{V}, 1 \leq t \leq ||\mathbf{w}||$
    - token $j$ in vocabulary $\mathbf{V}$ is denoted $\mathbf{V}_j$
- We define two pseuduo-tokens to denote the start/end of the sentence
    - $\mathbf{w}_(0) = \text{<START>}$
    - $\mathbf{w}_{(n_{\mathbf{w}}+1)} = \text{<END>}$

We need a function to convert a token into a numeric vector:

$$\text{rep} : \text{token} \mapsto \mathbb{R}^{n_{\mathbf{V}}}$$

One Hot Encoding (OHE) and word embeddings are examples of such a function.

- For OHE: $n_{\mathbf{V}} = ||\mathbf{V}||$
- For Word Embeddings: $n_{\mathbf{V}}$ is the dimension of the embedding vector

We will extend $\text{rep}$ to sequences $\mathbf{w}$:

$$\text{rep}(\mathbf{w}) = \left[\text{rep}(\mathbf{w}_{(t)}) | 1 \le t \le ||\mathbf{w}||\right]$$

## Issue 1: Words are categorical variables

We address the first issue relating to text: words are *categorical variables*.

By now, we should know to **not** treat categorical variables as ordinals.

Let's review the reason.

Treating a word as an ordinal

$$\text{rep}(w) \in \mathbb{R}^1$$

would imply

- "apple" < "orange" is a sensible statement
- that this ordering is meaningful to a Machine Learning model

## Example

Linear regression:

$$\mathbf{y} = \Theta^T \mathrm{rep}(w)$$

Predict $\mathbf{y}$ given feature vector (attributes) $\mathrm{rep}(w)$

- by learning parameters $\boldsymbol{\Theta}$

Suppose that we tried to encode word $w$ with an integer: $\text{rep}(w) = I_w$.

- $I_{\text{apple}} = 10 * I_{\text{orange}}$
    - means "apple" has 10 times the impact on prediction $\hat{\mathbf{y}}$ as "orange"
    - impact is $\Theta * I_w$
- Re-encoding "apple" with a value 10 times larger would make it 10 times more important

## Sparse Represention of words by One Hot Encoding (OHE)

So the natural way of representing a word is as a categorical variable

- indictor per word: $\mathrm{Is}_{\mathrm{apple}}$
- One Hot Encoding

OHE is a *sparse* representation

- length of $\text{rep}(w)$ is $|\mathbf{V}|$, yet only a single non-zero element

The problem is that there are lots of words !

- $|V|$ is large !
- $\text{rep}(\mathbf{w})$ length is $|\mathbf{w}||\mathbf{V}|$

## Issue 2: Word-streams are variable length

We are already familiar with two ways of dealing with variable length input

- Use a Recurrent model, which handles sequences of arbitrary length
- Convert to a fixed length representation using pooling

## Fixed length representation via pooling

One way to deal with a sequence $\mathbf{w}$ of words is to convert it to a vector of **fixed length**.

Once the length is fixed

- Classical and Deep Learning models taking fixed length inputs can work as usual.

Doing so usually involves losing the ordering information.

## Bag of Words (BOW): Pooling

We define a *reduction* operation $\mathrm{CBOW}$

- convert a sequence $\mathbf{w}$ of $||\mathbf{w}||$ elements
- each element of length $||\mathrm{rep}(\mathbf{w}_{(t)})||$
- to a fixed length vector of length $||\mathrm{rep}(\mathbf{w}_{(t)})||$

This will necessarily lose token order: this method is called *Bag of Words (BOW)*

There are many operators to achieve the reduction, which we will group under the name *pooling*

## Sum/Average

$$\text{CBOW}(\mathbf{w}) = \sum_{t=1}^{||\mathbf{w}||} \text{rep}(\mathbf{w}_{(t)})$$

Since $\mathbf{w}_{(t)}$ is a vector, the addition operation is element-wise.

So the composite vector for the sequence is the sum of the vectors of each element in the sequence.

We can easily turn the Sum into an average by dividing by $||\mathbf{w}||$

**Count vectorization:**

In the special case that

$$\mathrm{rep}(\mathbf{w}_{(t)}) = \mathrm{OHE}(\mathbf{w}_{(t)})$$

$\mathrm{CBOW}(\mathbf{w})_j$ is equal to the number of occurrences in sequence $\mathbf{w}$ of the $j^{th}$ word in $\mathbf{V}$.

This is often called *Count Vectorization*.

## TF-IDF

Count Vectorization is simple but ignores a basic fact or language

- word "importance" is often inversely correlated with frequency in $\mathbf{V}$

In English:

- the words "a", "the" and "is" are extremely high frequency (so high counts in most $\mathbf{w}$).
- but are so common as to convey little meaning

On the other hand, a rare word (or sequence of words) may be very distinctive ("Machine Learning").

*Term Frequency, Inverse Document Frequency (TF-IDF)*

- is based on the idea that a word that is *infequent* in the wide corpus
- but is frequent in a particular document in the corpus is very meaningul in the context of the document.

So a document

- in which "Machine Learning" occured a disproportionately high (relative to the broad corpus) number of times
- is likely to indicate that the document is dealing with the subject of Machine Learning.

**Note** A similar idea is behind many Web search algorithms (Google).

TF-IDF is similar to the Count Vectorizer, but with modified counts that are the product of

- the frequency of a word within a single document
- the inverse of the frequency of the word relative to all documents
- $v$ is a word
- $d$ is a document (collection of words) in set of documents $D$

$$\begin{aligned}
\text{tf}(v, d) &= \text{frequency of word } v \text{ in document } d && \text{(Term Frquency)} \\
\text{df}(v) &= \text{number of documents that contain word } v \\
\text{idf}(v) &= \log(\frac{||D||}{\text{df}(v)}) + 1 && \text{Inverse Document F} \\
\\
\text{tf-idf}(v, d) &= \text{tf}(v, d) * \text{idf}(v)
\end{aligned}$$

## Detour: Sentiment classification notebook on Colab : simple model

Classification task

- Input: Movie review, as sequence of characters
- Label: Positive/Negative

NLP notebook: examine the data (https://colab.research.google.com/github/kenperry-public/ML_Fall_2019/blob/master/Keras_examples_imdb_cnn.ipynb#scrollTo=shHO2IU8

NLP notebook: simple model (https://colab.research.google.com/drive/15KZrB_qR63Q3KjLVdaT2BV-NgsdcXo4F#scrollTo=QtvUFJZJ7Oqi)

## Back from the detour: summary of SImple Model

- One Hot Encoded words
    - OHE via an Embedding layer
        - Use Embedding as pedagogical device; world's *slowest* way to perform OHE !
- Variable length sequence of words
- Global Pooling to reduce to fixed length

## Issue 3: Ordering matters, first attempt using Convolution

Not every text problem requires the complete ordering of words.

We will briefly discuss non-ordered methods of dealing with text.

## Neural n-grams using Conv1d

An *n-gram* is a sequence of $n$ consecutive tokens that encapsulates a single concept (*phrase*)

An n-gram captures

- multi-token concept
  - "New York City" versus [ "New", "York", "City" ]
- ordering information
  - [ "hard", "not", "easy" ] versus [ "easy", "not", "hard" ]

NLP can be enhanced by replacing the subsequence of related words by the n-gram.

How does one identify n-grams ? There are two approaches.

The first is statistical

- joint frequency of the phrase's tokens being higher than the product assuming independence
- $p(\text{"New York City"}) > p(\text{"New"})p(\text{"York"})p(\text{"City"})$

That is: the frequency of the phrase is greater than the joint probability of its components, assuming independence.

The other way is: use Machine Learning !

- Discover consecutive $n$ tokens that are useful for some task

Using one dimensional convolution with kernel size $n$

- the convolution encodes each group of $n$ consecutive tokens (assuming stride 1)
- using multiple kernels: we can create an n-gram per kernel that captures some concept

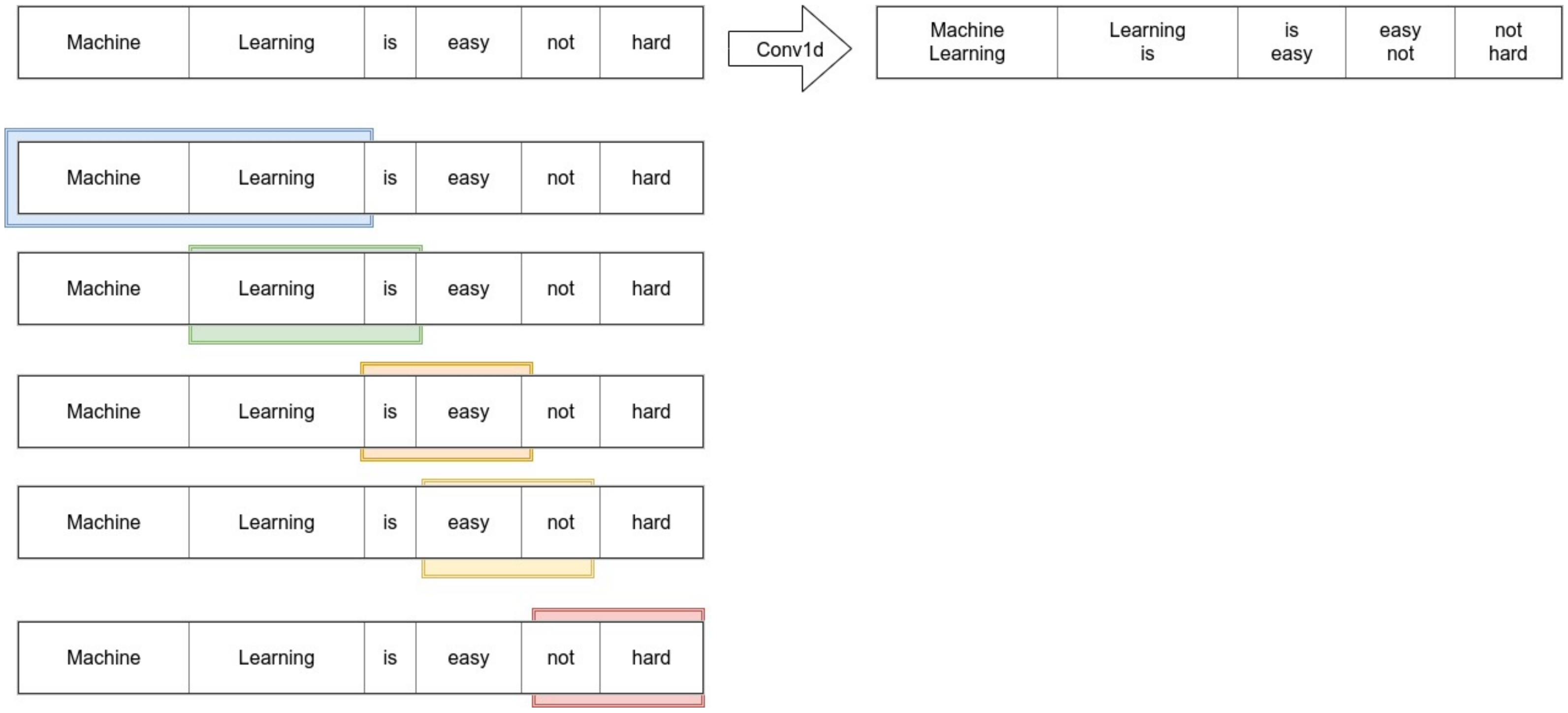
That is: we have created a new feature, per kernel, at each location of the text sequence.

n-grams can capture partial ordering of words (within span $n$).

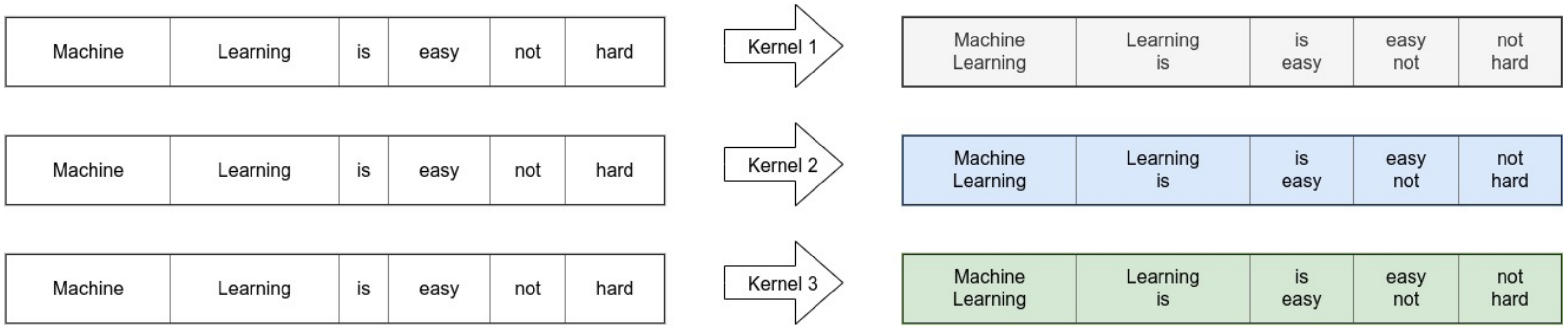
So creating n-grams (with varying $n$)

- before applying Pooling
- retains local ordering for features within span of $n$ tokens

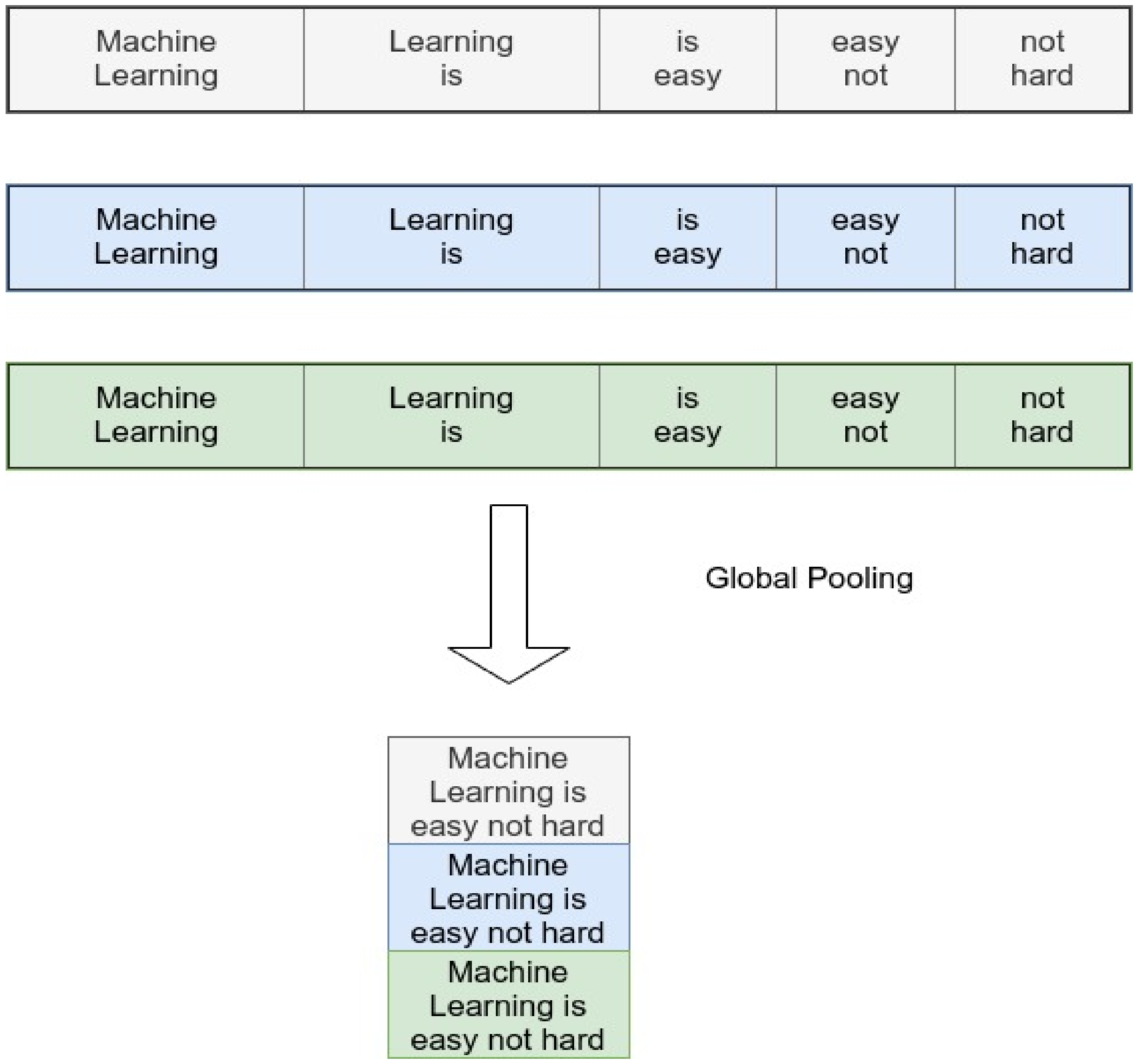
## One dimensional convolution



## One dimensional convolution, multiple kernels



## Global Pooling



## Detour: Sentiment classification notebook on Colab : Neural n-grams

NLP notebook: neural n-grams (https://colab.research.google.com/github/kenperry-public/ML_Fall_2019/blob/master/Keras_examples_imdb_cnn.ipynb#scrollTo=LPChvdnzU

## Back from the detour: summary of Neural n-grams

- One dimensional convolution over time dimension
    - 3-grams
- Global Pooling

## Issue 1 revisited: Sparse verus dense representation of categoricals

### Dense representation of words: Embeddings

Sparse encodings, such as OHE

- convert a token into a vector of features
- where the features are orthogonal: only one is active at a time

This is called a *discrete* representation.

Discrete representations have a major drawbacks

- they are long
    - $\text{rep}(\mathbf{w})$ length is $||\mathbf{w}|| * ||\mathbf{V}||$
- there is no meaningful metric of "distance" between the representation of words

To illustrate the "lack of distance" issue, let

$$\text{OHE}(w)$$

denote the One Hot Encoding of word $w$.

Using dot product (cosine similarity) as a measure of similarity

| **word** | **OHE(word)** | **Similarity** |
|---|---|---|
| dog | [1,0,0,0] | OHE(word) · OHE(dog) = 1 |
| dogs | [0,1,0,0] | OHE(word) · OHE(dog) = 0 |
| cat | [0,0,1,0] | OHE(word) · OHE(dog) = 0 |
| apple | [0,0,0,1] | OHE(word) · OHE(dog) = 0 |

Each pair of distinct words has 0 similarity

- no recognition of plural form
- no recognition of commonality (pets)

This is due to the fact that only a single "feature" of the OHE is active (non-zero).

However, it's possible that, in reality, there are many "dimensions" to a word, for example

- singular/plural
- entity type, e.g., Person
- positive/negative

- "Cats", "Dogs", "Apples"
    - related by being plural form
- "Cat", "Dog"
    - related by being animals
- "good", "bad"
    - related by being "opposites"

Thus it is not unreasonable to represent a word as a short *dense vector* of features

- each feature (vector element) captures a concept
- numeric value of element encodes the strength of the word's relation to the concept

Ideally the features would be indepenent

This is called a *continuous* word representation.

## Doing math with words

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

Let $\mathbf{v}_w$ be the dense vector/embedding for word $w$

- captures multiple aspects of a word
- where each element of the vector is a nearly-independent aspect
- then we can perform interesting mathematical manipulations on word vectors

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

Does the last dimension encode "plural form" ?

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

If so:

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

## Word analogies

king:man :: ? : queen

Let

- $\mathbf{v}_w$ be the dense vector for word $w$
- $d(\mathbf{v}_w, \mathbf{v}_{w'})$ be some measure of the distance between the two vectors $\mathbf{v}_w, \mathbf{v}_{w'}$
    - e.g., ( $1 - \text{cosine similarity}$ )

Using the distance metric, define the set of words in vocabulary $\mathbf{V}$ that are "closest" to a word $w$.

Let

- $\text{wv}_{n',d}(\mathbf{v}_w)$ be the dense vectors of the $n'$ words in $\mathbf{V}$ closest to word $w$
$$\text{wv}_{n',d}(\mathbf{v}_w) = \{\mathbf{v}_{w'} | \text{rank}_V(d(\mathbf{v}_w, \mathbf{v}_{w'})) \leq n'\}$$
- $N_{n',d}(w)$ be the set of $n'$ words associated with $\text{wv}_{n',d}(\mathbf{v}_w)$
$$N_{n',d}(w) = \{w' | w' \in \text{wv}_{n',d}(\mathbf{v}_w)\}$$

We can define approximate equality of two words $w, w'$ if they are among the closest words

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

That is:

- word $w$ is approximately equal to word $w'$
- if $w'$ is among the $n'$ words closest to $w$ according to distance metric $d$.

Finally, we can define word analogies:

a:b :: c:d

means

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

So to solve the word analogy for $c$:

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

To be concrete:

$$\mathbf{v}_{\text{king}} - \mathbf{v}_{\text{man}} + \mathbf{v}_{\text{woman}} \approx_{n',d} \mathbf{v}_{\text{queen}}$$

## Why does adding 2 word vectors work

- Mikolov
    - Vector for a word reflects its context
    - Vector is log probability
        - so sum of log probabilities is log of product of probabilities
            - product is like a logical "and"

## 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 corpra

- 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 $100$.

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

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

You may discover some unintended bias

$$\begin{aligned}
\text{doctor - man + woman} &\approx_{n',d} \text{nurse} \\
\text{mechanic - man + woman} &\approx_{n',d} \text{teacher}
\end{aligned}$$

## Domain specific embeddings

Do we speak Wikipedia English in this room ?

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

$$\begin{aligned}
N(\text{bull}) &= [\text{cow, elephant, dog, wolf, pit, bear, rider, lion, horse}] \\
N(\text{short}) &= [\text{rather, instead, making, time, though, well, longer, shorter, long}] \\
N(\text{strike}) &= [\text{workers, struck, action, blow, striking, protest, stoppage, walkout,} \\
N(\text{FX}) &= [\text{showtime, cnbc, ff, nickelodeon, hbo, wb, cw, vh1}]
\end{aligned}$$

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 representation of words ?

We learn them !

Suppose we had a task T that involves mapping a sequence of words to an outcome.

To be concrete: mapping a movie review to an indicator of Positive/Negative sentiment.

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

- inputs are OHE of words
- target is Positive/Negative label
- Logistic Regression from sentence representation to binary target Positive/Negative

One could also ask

- can we map the OHE of a word $\mathbf{w}_{(t)}$ (length $|\mathbf{V}|$)
- to a shorter, dense vector $\mathbf{e}_{(t)}$ of length $n_e$
- and use the dense vector in the Logistic Regerssion

This mapping can be represented by an an $(|\mathbf{V}| \times n_e)$ matrix $\mathbf{E}$

$$\mathbf{e}_{(t)} = \text{OHE}(\mathbf{w}_{(t)})^T \mathbf{E}$$

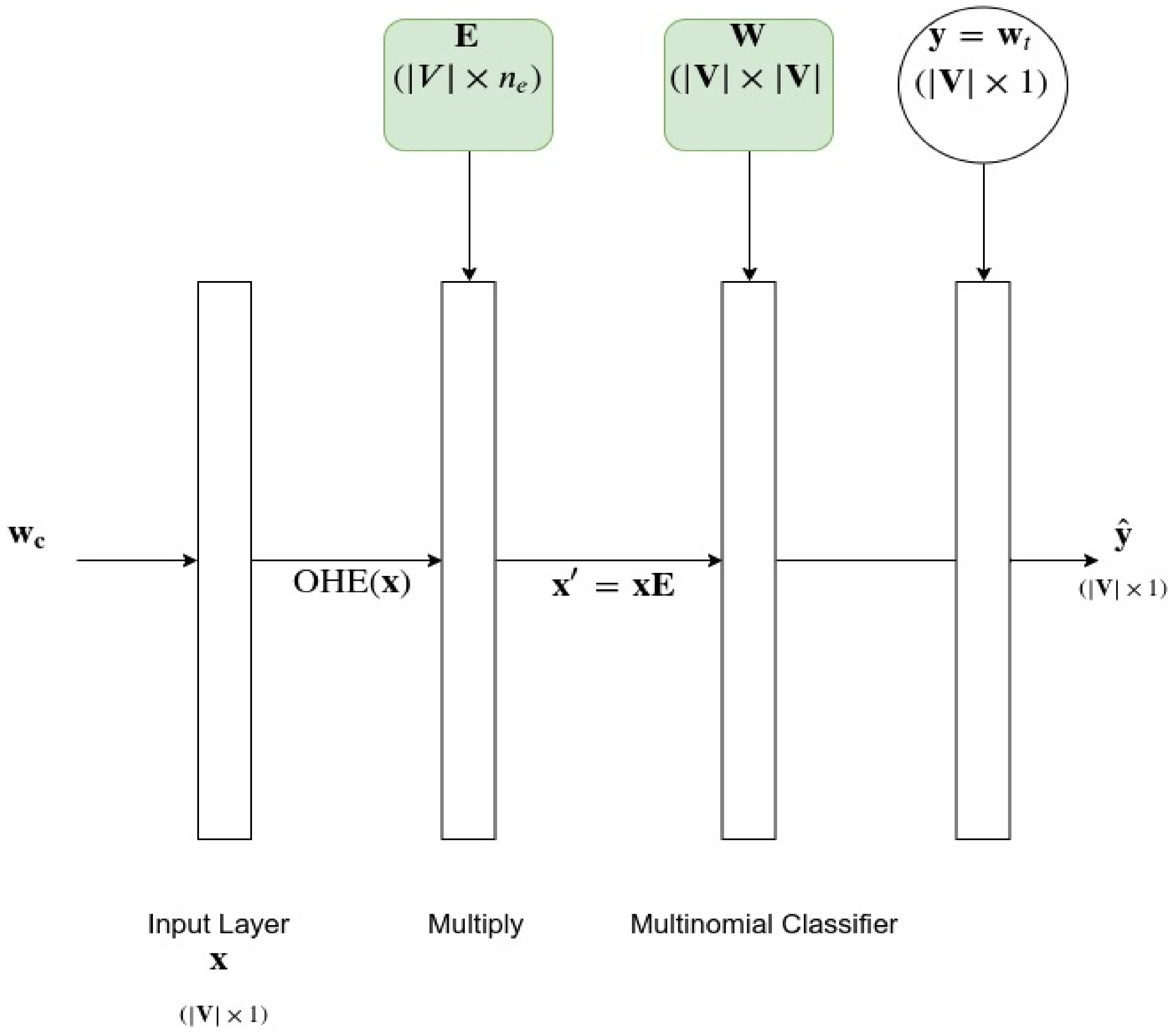
Using Machine Learning,

- we solve for both the Logistic Regression parameters $\mathbf{W}$ *and* $\mathbf{E}$
- when solving the Classification Task via Logistic Regression.

The matrix $\mathbf{E}$ is called

- an *embedding matrix* for words
- and $\mathbf{e}_{(t)}$ is called an *embedding* or *word vector* for word $\mathbf{w}_{(t)}$.

*Word embeddings* have become an important component of Deep Learning for NLP.


$\mathbf{E}$
$(|V| \times n_e)$
$\mathbf{W}$
$(|\mathbf{V}| \times |\mathbf{V}|$
$\mathbf{y} = \mathbf{w}_t$
$(|\mathbf{V}| \times 1)$
$\mathbf{w_c}$
$\mathrm{OHE}(\mathbf{x})$
$\mathbf{x}' = \mathbf{x}\mathbf{E}$
$\hat{\mathbf{y}}$
$(|\mathbf{V}| \times 1)$
Input Layer
$\mathbf{x}$
$(|\mathbf{V}| \times 1)$
Multiply
Multinomial Classifier

In other words

- we have learned a dense vector representation of words $\mathbf{E}$
- that is useful for a particular classification task

Might it be possible that the dense vector representation of words for this task

- is useful for other tasks involving words ?
- this is Transfer Learning

The problem with this approach is having a large enough training set for the task $T$.

We will show how to solve this problem using semi-supervised learning

- word prediction problems

## Word prediction problems: high-level

Let's explore how to create generally useful (as opposed to task specific) word embeddings.

In the absence of labelled data (needed for Supervised Learning)

- we can create a Semi-Supervised Learning task

From unlabelled sequence $\mathbf{w}$ define the *word prediction* problem as

- predict a target word given a "context" sequence of words

For example:

- given prefix $\mathbf{w}_{(1)}$
$\ldots \mathbf{w}_{(t-1)}$
- predict $\mathbf{w}_{(t)}$.

The inspiration is that if you can predict the occurrence of word from it's neighbors that you have somehow capture dimensions of meaning.

This is often refered to as "a word is known by the company it keeps".

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

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

Word embeddings can be obtained as a by-product of this *word prediction* problem.

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 | $p(\mathbf{w}_{(t)}\vert$ | $\mathbf{w}_{(t-o)} \dots , \mathbf{w}_{(t-1)})$ |
| predict a surrounding word | $p(\mathbf{w}_{(t')}\vert$ | $\mathbf{w}_{(t)})$ |
| | | $t' = \{t-o, \dots , t+o\} - \{t\}$ |
| predict center word from context | $p(\mathbf{w}_{(t)}\vert$ | $[\mathbf{w}_{(t-o)} \dots \mathbf{w}_{(t-1)}\mathbf{w}_{(t+1)} \dots \mathbf{w}_{(t+o)}])$ |

# Word prediction problems, in detail

## Background: Language models

A *Language Model* takes a sequence of words and produces a *probability* that the sequence represents a sentence in the language.

- We will show how we can obtain this probability by predicting the probability of word $\mathbf{w}_{(t)}$ conditional on the first $(t-1)$ words in the sequence.
- We will then simplify this by a problem that involves predicting word $w_{(t)}$ from a *small* window in the neighborhood of word $t$.

Two variants of the window-based approach are the basis for a popular word embedding techinique: word2vec.

Let $\mathbf{w}$ be the sequence of $n$ words in a sentence.

A *language model*

- maps $\mathbf{w}$ into a probability $p(\mathbf{w})$ that $\mathbf{w}$ represents a sentence in the language.

We can compute this probability via the chained probabilitiy

$$p(\mathbf{w}) = p(\mathbf{w}_{(1)}|\mathbf{w}_{(0)})\ p(\mathbf{w}_{(2)}|\mathbf{w}_{(0)}\mathbf{w}(1)) \ldots\ p(\mathbf{w}_{(n_{\mathbf{w}}+1)}|\mathbf{w}_{(0)} \ldots \mathbf{w}_{(n_{\mathbf{w}})})$$

or

$$p(\mathbf{w}) = \prod_{t=1}^{n_{\mathbf{w}}+1} p(\mathbf{w}_{(t)}|\mathbf{w}_{(0)} \ldots \mathbf{w}_{(t-1)})$$

That is

- the probablility of $\mathbf{w}$ be a valid sequence of tokens ("sentence")
- is the chained probability of token $\mathbf{w}_{(t)}$ following prefix $\mathbf{w}_{(0)} \ldots \mathbf{w}_{(t-1)}$
    - for each $1 \le t \le n_{\mathbf{w}}$

We can determine these probabilities via a *maximum likelihood* estimate by counting word sequence occurrences in our text corpus.

$$p(\mathbf{w}_{(t)} | \mathbf{w}_{(0)} \ldots \mathbf{w}_{(t-1)}) = \frac{\text{count}_{\text{Corpus}} \mathbf{w}_0 \cdots \mathbf{w}_{(t)}}{\text{count}_{\text{Corpus}} \mathbf{w}_0 \cdots \mathbf{w}_{(t-1)}}$$

The estimate via counting is an unrealistic ideal

- we don't have *all* possible sentences in the language; the corpus is a subset
- may not have *true* probability for rare words in language when corpus is small
- computationally expensive
- Out of Vocabulary (OOV) problem
    - tokens appearing at inference (test) time that were *not* in training

## Window based models

It is more realistic to *approximate*

$$p(\mathbf{w}_{(t)}|\mathbf{w}_{(0)} \ldots \mathbf{w}_{(t-1)})$$

by conditioning $\mathbf{w}_{(t)}$ on a **fixed length** prefix ending at $t-1$

- Unigram (1-gram) approximation $p(\mathbf{w}_{(t)}|\mathbf{w}_{(0)} \ldots \mathbf{w}_{(t-1)}) \approx p(\mathbf{w}_{(t)})$
    - That is, conditional probability of $\mathbf{w}_i$ is just the unconditional probability.
- Bigram (2-gram) approximation $p(\mathbf{w}_{(t)}|\mathbf{w}_{(0)} \ldots \mathbf{w}_{(t-1)}) \approx p(\mathbf{w}_{(t)}|\mathbf{w}_{(t-1)})$
- n-gram approximation $p(\mathbf{w}_{(t)}|\mathbf{w}_{(0)} \ldots \mathbf{w}_{(t-1)}) \approx p(\mathbf{w}_i|\mathbf{w}_{(t-(n-1))} \ldots \mathbf{w}_{(t-1)})$

You can probably see the weakness of the unigram model

- Doesn't respect word order
  $p(["\text{New}", "\text{York}"]) = p(["\text{York}", "\text{New}"])$

and how increasing the window improves the approximation.

The assumption that I can predict solely based on a prefix is called a *Markov* assumption.

## Word prediction problem for word2vec

word2vec is based on one of two prediction problems.

- predict center word given surrounding words as context
- precict which words can occur on either side of a given center words

The problems are framed as: Predict target word $w_t$ given conditional word $w_c$

- $p(w_t|w_c)$

The first prediction problems is called *Skip gram*: predict surrounding words

- conditional word is a "center word" that is surrounded by other words: $w_c = \mathbf{w}_{(t)}$
- target word is any surrounding word at a position $t'$ within a window of $t$
    - predict probability of any word $w_t \in \mathbf{V}$ being equal to $\mathbf{w}_{(t')}$
    - where $t' = \{t - o, \ldots, t + o\} - \{t\}$

The set of training examples (example/label pairs) associates with $w_c = \mathbf{w}_{(t)}$

$$\{(\mathbf{w}_{(t)}, \mathbf{w}_{(t')})|\ t' \in \{t - o, \ldots, t + o\} - \{t\}\}$$

The second prediction problem is called *CBOW*

- center word $\mathbf{w}_{(t)}$
- conditional word $w_c$ is any word that surrounds $\mathbf{w}_{(t)}$
- target word is the center word $w_t = \mathbf{w}_{(t)}$
    - predict probability of $w_t = w_{(t)}$ is center word
    - given a surrounding word $w_c \in \{\mathbf{w}_{(t')} | t' = \{t - o, \ldots, t + o\} - \{t\}\}$

The set of training examples (example/label pairs) associates with taarget $\mathbf{w}_{(t)}$

$$\{(\mathbf{w}_{(t')}, \mathbf{w}_{(t)}) |\ t' = \{t - o, \ldots, t + o\} - \{t\}\}$$

## word2vec derivation

Prediction problem as multinomial (one class per word in vocabulary) classification problem

$$\mathbf{y} = \mathbf{W}\mathbf{x}'$$

where

- $\mathbf{y}$ is a (OHE) target word
- $\mathbf{x}'$ is derived from (one or more) conditional words.
- $\mathbf{W}$ is $(||V|| \times ||V||)$
    - $\mathbf{W}^{\mathbf{(i)}}$, denoting row $i$ of $\mathbf{W}$ is the "template" for target word $\mathbf{V}_i$
        - there are $||V||$ such targets, one per word in $\mathbf{V}$
            - $\mathbf{W}^{\mathbf{(i)}}$ is length $||V||$, the size of the OHE

We want to obtain an embedding matrix $\mathbf{E}$ of dimension $(|\mathbf{V}| \times n_e)$

- $\mathbf{E}^{(j)}$, the $j^{th}$ row of $\mathbf{E}$ is the dense vector of $\mathbf{V}_j$, the $j^{th}$ word in vocabulary $\mathbf{V}$.
- so $x' = o * \mathbf{E}$
  - where $o = \text{OHE}(\mathbf{x})$
  - $x'$ is now length $n_e$ rather than $||V||$

The regression solves for both $\mathbf{W}$ (as usual) *and* $\mathbf{E}$.

That is: we find the embedding $\mathbf{E}$ that is best suited for the classification regression.

The matrix $\mathbf{E}$ would then be a map from word $\mathbf{V}_j$ to embedding $\mathbf{e}_j$.

This embedding matrix would hopefully "transfer" to other NLP tasks.

Let's not overlook $\mathbf{W}$, the matrix of classifier parameters

- $\mathbf{W}^{(j)}$, row $j$ of $\mathbf{W}$, is the "template" for target word $\mathbf{V}_j$
    - multinomial regression has one target per vocabulary word
    - it produces a probability distribution $\hat{\mathbf{y}}$ over the words in $\mathbf{V}$
    - $\hat{\mathbf{y}}_j$ is the probability associated with word $\mathbf{V}_j$
    - length of $\mathbf{W}^{(j)}$ is $n_e$, same as embedding vector

In some sense, $\mathbf{W}^{(j)}$ can *also* be thought of as a dense representation of $\mathbf{V}_i$

- $\mathbf{E}^{(j)}$ is representation of $\mathbf{V}_j$ when it is a conditional word (independent variable)
- $\mathbf{W}^{(j)}$ is representation of $\mathbf{V}_j$ when it is a target word (dependent variable)
    - i.e., is a template for target $\mathbf{V}_j$

It is usually the case, for simplicity

- to use average of $\mathbf{E}$ and $\mathbf{W}$ (which are the same size) as embedding

## Objective function

Maximize average log probability over the $T$ examples in training set:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{t'\in\{t-o,\ldots,t+o\}-\{t\}}\log(p(w_{(t')}|w_{(t)}))$$

## Detour: Sentiment classification notebook on Colab : Learned embeddings

NLP notebook: learned embeddings
(https://colab.research.google.com/github/kenperry-
public/ML_Fall_2019/blob/master/Keras_examples_imdb_cnn.ipynb#scrollTo=f5XrUD3X8

## Back from the detour: summary of Learned embeddings

## Issues 2,3 revisited: Variable length, ordered token sequences

We should already have some idea of how to deal with variable length sequences.

Recurrent Neural Networks take sequence inputs and create representations (hidden states) that are fixed length "encodings".

Recall that the RNN produces a sequence of hidden states $\mathbf{h}(0), \ldots, \mathbf{h}_{(||\mathbf{w}||)}$.

Hidden state $\mathbf{h}_{(t)}$ is effectively a summary or encoding of $\mathbf{w}_{(0)}\mathbf{w}_{(1)} \ldots \mathbf{w}_{(t)}$

- it is computed after having seen the prefix of $\mathbf{w}$ ending at $t$.

So $\mathbf{h}_{(t)}$ is a fixed length encoding of $\mathbf{w}_{(0)}\mathbf{w}_{(1)} \ldots \mathbf{w}_{(t)}$.

This gives us a way to convert variable length $\mathbf{w}$ into fixed length $\mathbf{h}_{(||\mathbf{w}||)}$

```
In [2]: print("Done")
```

```
Done
```