Text Encoding Methods

In Natural Language Processing (NLP), encoding text is essential because computers cannot directly understand human language in its raw form.

Text must be converted into **numerical representations that algorithms can process and analyze**.

This transformation allows models to capture the structure, meaning, and context of language, enabling tasks like sentiment analysis, machine translation, and text classification.

Text encoding bridges the gap between human communication and machine understanding.

Commonly used encoding methods

Encoding Method	Overview	Common Uses
Binary / One-Hot Encoding	Records presence (1) or absence (0) of words in a document.	Basic text classification, spam detection
Count / Frequency Encoding	Counts how often each word appears in a document.	Document similarity, naive Bayes classifiers
TF-IDF (Term Frequency-Inverse Document Frequency)	Weighs word frequency against how common the word is across all documents.	Information retrieval, keyword extraction, text classification
Word2Vec	Maps words to dense vectors based on context in large corpora.	Word similarity, sentiment analysis, embeddings for deep models
Transformer-based Encoding (e.g., BERT)	Generates contextualized embeddings that consider word meaning based on sentence context.	Question answering, text summarization, language understanding

Differences between **encoding** and **embedding**:

Aspect	Encoding	Embedding
Definition	General method to convert text into numerical format	Dense vector representation capturing semantic meaning
Type	Broad category (includes one-hot, TF-IDF, etc.)	Specific type of encoding (dense and learned)
Density	Often **sparse** (many zeros)	Always **dense** (compact vector)
Dimensionality	Can be **very high** (e.g., one-hot)	**Lower-dimensional** and fixed (e.g., 300-d for Word2Vec)
Learning Method	Typically **rule-based or count-based**	**Learned from data**, often using neural networks
Semantic Awareness	Usually **lacks semantic context**	**Captures semantic and contextual meaning**
Examples	One-hot, Bag of Words, TF-IDF	Word2Vec, GloVe, BERT, FastText
Use Cases	Simple models, baseline NLP tasks	Deep learning models, semantic search, NLP understanding

Examples: Original text and its encodings / embeddings:

```
"the class started with a hands on demo. sir uploaded a file on
moodle. we then downloaded the file and applied simple linear
regression on it. sir then explained about some features in excel
and how to use them. sir then focused on errors in linear
regression. then sir explained about histogram. outcome dependent
on a large number of unknown causes(random), the distribution is
gaussian normal distribution.
we need to understand what each number we get from analysis tells
us. lower and upper 95% means that the real value of population
lies with 95% confidence in the interval [lower 95,upper 95].
what is a good model- one that explains most of the variations in
the data.
sst=sum of (yi-y_bar)â² (measure of total variation in given
dataset)
sst=sse+ssr
ssr- sum of square of regression line (total variation explained
by the regression line)
sse- variation not explained by the model, attributed to random
errors.
sst=ssr+sse
1=(ssr/sst)+sse/sst
ssr/sst portion of total variation described by the regression
model. for a nice model we need this value to be as high as
```

```
Count / Frequency Encoding

"[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 2 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 2 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 5
 0 0 0 0 0 0 0 0 0 0 0 0 1 2 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0
 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0
 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 2 3 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 1 0 0
 0 0 0 0 0 2 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
TF-IDF Encoding

"[0.         0.         0.         0.         0.         0
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.03488096 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.0
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.08105591 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.10815726 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.136909   0.
 0.         0.         0.         0.         0.0584522  0.
```

```
word2vec Embedding

"[-9.8596076e-03  3.8282499e-02  4.6141213e-04  1.8042079e-03
  1.7823273e-02 -8.4031224e-02  2.4463106e-02  1.0235916e-01
 -3.8112249e-02 -2.6500074e-02 -3.3631958e-02 -7.0363007e-02
  9.9193342e-03  1.7697796e-02  1.9611845e-02 -2.6358224e-02
  9.7975479e-03 -5.0550874e-02 -3.2687932e-03 -9.6165709e-02
  3.0920580e-02  2.5498468e-02  1.9252919e-02 -2.2812249e-02
 -1.0838713e-02  1.6571671e-02 -4.6286643e-02 -2.7317706e-02
 -3.3590551e-02  1.6243888e-04  5.4744702e-02  1.0432001e-02
  3.2210767e-02 -2.9655201e-02 -2.6723975e-02  6.0300447e-02
 -3.4390076e-04 -3.3453923e-02 -3.2798301e-02 -9.4663821e-02
  4.0563056e-03 -3.9952219e-02 -3.1474914e-02  5.3761603e-04
  4.3891892e-02 -1.5204668e-04 -3.7732240e-02 -1.7385455e-02
  3.8724329e-02  2.0032451e-02  1.8899508e-02 -3.3062652e-02
 -9.5166289e-04 -1.3119498e-02 -2.0647902e-02  2.1207543e-02
  1.0926923e-02  3.1357707e-05 -5.3577535e-02  1.2675912e-02
  1.1813557e-02  4.7166348e-03  3.7921940e-03 -1.2941050e-03
 -5.1513478e-02  3.8718920e-02  2.3156255e-02  3.7282098e-02
```

Note:

- Text encodings / embeddings are ***vectors***
- Count and TF-IDF encodings result in ***sparse vectors***
- word2vec embedding results in a ***dense vectors***

Steps leading to encodings/embeddings (Traditional NLP)

Step	Description
1. Raw Text Input	Unprocessed text.
2. Text Cleaning	Lowercasing, remove punctuation/special characters.
3. Stopword Removal	Filter out frequent but uninformative words.
4. Tokenization	Split into tokens (if not already).
5. Normalization	Apply stemming or lemmatization to tokens.
6. Vocabulary Construction	Map tokens to IDs.
7. Encoding / Embedding	Convert to numerical form.

In **deep learning or transformer-based models**, **stopwords are often kept**, because:

- Models learn to weigh their importance (or lack thereof).
- Context matters, even for stopwords.

Steps to Create Sentence or Document Embeddings from Word Embeddings

Step	Description
1. Create word embeddings	Follow the steps for word embeddings
2. Word Embedding Lookup	For each token, fetch its corresponding embedding (e.g., from Word2Vec, GloVe, etc.).
3. Combination Strategy	Combine the word vectors into a single vector for the sentence or document. Common methods include:
• **Averaging**	Take the mean of all word embeddings — simple and fast.
• **Sum / Weighted Sum**	Sum embeddings or use weights (e.g., TF-IDF) for importance.
• **Max / Min Pooling**	Take max or min values across all word vectors (captures strong features).
• **Concatenation**	Combine word embeddings for selected positions (rarely scalable for long text).
4. Post-processing (Optional)	Normalize the resulting vector, reduce dimensions (e.g., PCA), or refine using sentence-level models.

Binary / one-hot encoding and Vocabulary Creation Example

```
Document 1: "Machine learning is fascinating."
Document 2: "I enjoy learning about machine learning techniques."
Document 3: "Deep learning models are a subset of machine learning."
```

CORPUS.

- Convert to lower case
- Drop the stop words & other punctuations
- Create the vocabulary - an indexed list of words

Vocabulary : {machine, learning, fascinating, enjoy, techniques, deep, models, subset}
0 1 2 3 4 5 6 7

Vector embeddings of the documents

d1: {machine, learning, fascinating} = {1,1,1,0,0,0,0,0} → this vector now represents the doc.

d2: {enjoy, learning, machine, techniques} = {machine, learning, enjoy, techniques} = {1,1,0,1,1,0,0,0}

d3: {deep, learning, models, subset, machine, learning} = {machine, learning, deep, models, subset} = {1,1,0.0.0.1.1.1}

Note :

- All documents are expressed in terms of the vocabulary
- All documents expressed in terms of vector of the same length

Applications:

- Distance between the documents can now be found out by using various measures of distance. Eg. Euclidean distance
- Similarity / difference between the documents can be established by using Cosine Similarity

Example: Distance:

$$d(d_1 d_2) = \sqrt{(1-1)^2+(1-1)^2+(1-0)^2+(0-1)^2+(0-1)^2+(0)^2+(0)^2+(0)^2} = \sqrt{0+0+1+1+1+0+0+0} = \sqrt{3}$$

$$= 1.732$$

$$d(d_2 d_3) = \sqrt{0+0+0+1+1+1+1+1} = \sqrt{5}$$

$$= 2.236$$

$$d(d_1 d_3) = \sqrt{0+0+1+0+0+1+1+1} = \sqrt{4}$$

$$= 2$$

Limitations: – Frequency of words in the document not captured in this scheme → solution → TF-IDF.

Calculation of cosine similarity using the encoding vectors

Vocabulary : {machine, learning, fascinating, enjoy, techniques, deep, models, subset}

0 1 2 3 4 5 6 7

Vector embeddings of the documents

d1: {machine, learning, fascinating} = {1,1,1,0,0,0,0,0} → this vector now represents the doc.

d2: {enjoy, learning, machine, techniques} = {machine, learning, enjoy, techniques} = {1,1,0,1,1,0,0,0}

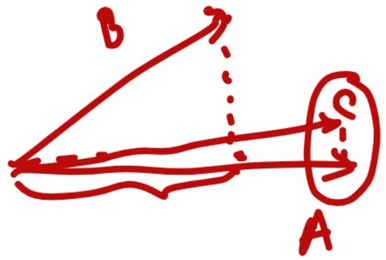
d3: {deep, learning, models, subset, machine, learning} = {machine, learning, deep, models, subset} = {1,1,0.0.0.1.1.1}

The **cosine similarity** between two vectors **A** and **B** is:

$$\text{cosine\_similarity}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|}$$

And **cosine distance** is:

$$\text{cosine\_distance}(A, B) = 1 - \text{cosine\_similarity}(A, B)$$



- **d1 · d2:**

$$1 \times 1 + 1 \times 1 + 1 \times 0 + 0 \times 1 + 0 \times 1 + 0 \times 0 + 0 \times 0 + 0 \times 0 = 2$$

- |d1| = √(1² + 1² + 1²) = √3 ≈ 1.732
- |d2| = √(1² + 1² + 1² + 1²) = √4 = 2

→ **cosine_sim(d1, d2)** = 2 / (1.732 × 2) ≈ 2 / 3.464 ≈ **0.577**

→ **cosine_dist(d1, d2)** = 1 - 0.577 = **0.423**

- **d1 · d3:**

$$1 \times 1 + 1 \times 1 + 1 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 1 + 0 \times 1 + 0 \times 1 = 2$$

- |d3| = √(1² + 1² + 1² + 1² + 1²) = √5 ≈ 2.236

→ **cosine_sim(d1, d3)** = 2 / (1.732 × 2.236) ≈ 2 / 3.873 ≈ **0.517**

→ **cosine_dist(d1, d3)** = 1 - 0.517 = **0.483**

- **d2 · d3:**

$$1 \times 1 + 1 \times 1 + 0 \times 0 + 1 \times 0 + 1 \times 0 + 0 \times 1 + 0 \times 1 + 0 \times 1 = 2$$

- |d2| = 2 (already computed)
- |d3| = 2.236 (already computed)

→ **cosine_sim(d2, d3)** = 2 / (2 × 2.236) ≈ 2 / 4.472 ≈ **0.447**

→ **cosine_dist(d2, d3)** = 1 - 0.447 = **0.553**

Other measures of Similarity and Distance

Cosine Similarity

- **Definition**: Measures the angle between two vectors, not their magnitude.
- **Formula**:
  Cosine Similarity = (d1 • d2) / (||d1|| * ||d2||)
  - d1 • d2 is the dot product of the two vectors
  - ||d1|| and ||d2|| are the magnitudes (lengths) of the vectors
- **Range**: -1 to 1
  - 1 = identical direction (most similar)
  - 0 = completely unrelated (orthogonal)
  - -1 = opposite direction
- **Use case**: Compare how similar two texts are based on their vector representations.

Cosine Distance

- **Definition**: Converts similarity into a distance metric.
- **Formula**:
  Cosine Distance = 1 - Cosine Similarity
- **Range**: Typically 0 to 2 (commonly seen between 0 and 1 when vectors are non-negative)
- **Use case**: Used in clustering or distance-based models where smaller values mean closer items.

TF-IDF (Term Frequency – Inverse Document Frequency)

TF-IDF is a statistical measure that evaluates how important a word is to a document in a collection of documents. It helps highlight important terms while down-weighting common words.

- **TF** highlights how important a term is **within a specific document**.
- **IDF** reduces the importance of terms that appear **frequently across the entire corpus** (i.e., common words).

Use of TF-IDF: ¶

- Text classification, document clustering
- Search engines (ranking based on term importance)
- Keyword extraction

1. **Term Frequency (TF):**

   Measures how often a word appears in a document relative to the total number of words in that document.

   Formula:

$$\text{TF}(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}$$

2. **Inverse Document Frequency (IDF):**

   Measures how common or rare a word is across all documents in the corpus. Words that appear in many documents get lower weights.

   Formula:

$$\text{IDF}(t) = \log\left(\frac{\text{Total number of documents}}{\text{Number of documents containing the term } t}\right)$$

3. **TF-IDF Score:**

   Combines both Term Frequency and Inverse Document Frequency to compute the final importance score for each word in the document.

   Formula:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

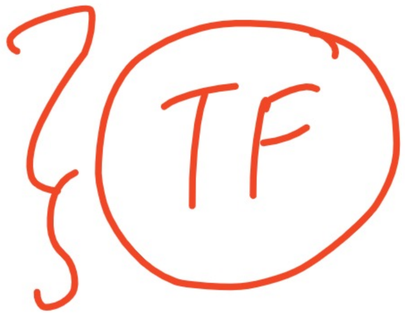
TF-IDF example

```
Document 1: "Machine learning is fascinating."
Document 2: "I enjoy learning about machine learning techniques."
Document 3: "Deep learning models are a subset of machine learning."
```

Vocabulary: {"machine", "learning", "fascinating", "enjoy", "techniques", "deep", "models", "subset"}

d1: {machine, learning, fascinating}
d2: {enjoy, learning, machine, learning, techniques}
d3: {deep, learning, models, subset, machine, learning}

TF Calculations
d1: {machine: 1/3, learning: 1/3, fascinating: 1/3}
d2: {enjoy: 1/5, learning: 2/5, machine: 1/5, techniques: 1/5}
d3: {deep: 1/6; learning: 2/6; models: 1/6; subset: 1/6; machine: 1/6}

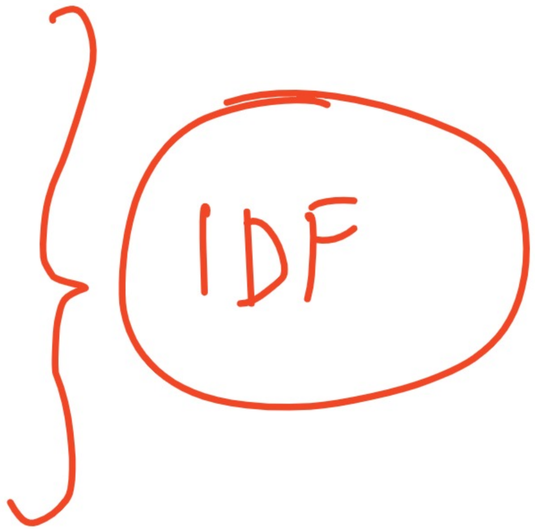


```
Document 1: "Machine learning is fascinating."
Document 2: "I enjoy learning about machine learning techniques."
Document 3: "Deep learning models are a subset of machine learning."
```

Vocabulary: {"machine", "learning", "fascinating", "enjoy", "techniques", "deep", "models", "subset"}

IDF Calculations for the vocabulary terms

- machine: log(3/3);
- learning: log(3/3);
- fascinating: log(3/1);
- enjoy: log(3/1);
- techniques: log(3/1);
- deep: log(3/1);
- models: log(3/1);
- subset: (3/1)



IDF-Vocabulary = {0, 0, 1.099, 1.099, 1.099, 1.099, 1.099, 1.099}

```
Document 1: "Machine learning is fascinating."
Document 2: "I enjoy learning about machine learning techniques."
Document 3: "Deep learning models are a subset of machine learning."
```

Vocabulary: {"machine", "learning", "fascinating", "enjoy", "techniques", "deep", "models", "subset"}

TF

d1: {machine: 1/3, learning: 1/3, fascinating: 1/3}
d2: {enjoy: 1/5, learning: 2/5, machine: 1/5, techniques: 1/5}
d3: {deep: 1/6; learning: 2/6; models: 1/6; subset: 1/6; machine: 1/6}

IDF

IDF-Vocabulary = {0, 0, 1.099, 1.099, 1.099, 1.099, 1.099, 1.099}

TFIDF Calculations and vector encodings for the documents

d1: {machine: 1/3*0; learning: 1/3*0; fascinating: 1/3*1.099; enjoy: 1/5*1.099; techniques: 1/5*1.099; deep: 0; models : 0; subset: 0) = {0,0,0.3663,0,0,0,0,0)

d2: {machine: 1/5*0; learning: 2/5*0; fascinating:0/5*1.099; enjoy: 1/5*1.099; techniques: 1/5*1.099; deep: 0; models : 0; subset: 0} = {0, 0, 0, 0.2198, 0.2198, 0, 0, 0)

d3: {machine: 1/6*0; learning: 2/6*0; fascinating:0/6*1.099; enjoy: 0/6*1.099; techniques: 0/6*1.099; deep: 1/6*1.099; models : 1/6*1.099; subset: 1/6*+1.099}
= {0, 0, 0, 0,0,0.1831, 0.1831,0.1831}

Distance and Similarity calculation: Based on TF-IDF vector embeddings of the documents:

d1 : {0,0,0.3663,0,0,0,0,0)

d2 : {0,0,0,0.2198, 0.2198,0,0,0)

d3 : {0,0,0,0,0,0.1831,0.1831,0.1831}

Pair: d1 & d2

Euclidean Distance:

$$\sqrt{(0.3663)^2 + (0.2198)^2 + (0.2198)^2} = \sqrt{0.1342 + 0.0483 + 0.0483} \approx \sqrt{0.2308} \approx 0.4804$$

Cosine Similarity: Dot product = 0 (no overlapping non-zero positions)

Magnitude of d1 = $\sqrt{(0.3663^2)} = 0.3663$

Magnitude of d2 = $\sqrt{(0.2198^2 + 0.2198^2)} = \sqrt{(0.0966)} \approx 0.3109$

$$\text{cos\_sim}(d1, d2) = 0/(0.3663 * 0.3109) = 0$$

Pair: d1 & d3

Euclidean Distance:

$$\sqrt{(0.3663)^2 + 3 \times (0.1831)^2} = \sqrt{0.1342 + 0.1006} = \sqrt{0.2348} \approx 0.4846$$

Cosine Similarity: Dot product = 0

Magnitude of d3 = $\sqrt{(3 \times 0.1831^2)} = \sqrt{(0.1006)} \approx 0.3172$

$$\text{cos\_sim}(d1, d3) = 0/(0.3663 \times 0.3172) = 0$$

Pair: d2 & d3

Euclidean Distance:

$$\sqrt{2 \times (0.2198)^2 + 3 \times (0.1831)^2} = \sqrt{0.0966 + 0.1006} = \sqrt{0.1972} \approx 0.4441$$

Cosine Similarity: Dot product = 0

$$\text{cos\_sim}(d2, d3) = 0/(0.3109 \times 0.3172) = 0$$

DISADVANTAGES : Contexts of the words are not considered; only their frequencies are considered.

Solution → Word2Vec.

The **word2vec** embedding method

Instead of treating words as discrete symbols (like one-hot vectors), Word2Vec learns **semantic relationships** between words by **predicting their context** in large text corpora.

Words that appear in **similar contexts** will have **similar vector representations**.

Instead of treating words as discrete symbols (like one-hot vectors), Word2Vec learns **semantic relationships** between words by **predicting their context** in large text corpora.

Words that appear in **similar contexts** will have **similar vector representations**.

Training Details

- **Input**: A massive corpus of text.
- **Output**: A matrix of word vectors (each row is a word embedding).
- **Mechanism**: Uses a shallow neural network trained via **negative sampling** or **hierarchical softmax**.
- **Window size**: Controls how many words before/after the target are considered context.
- **Vector size**: Controls the embedding vector size

Properties of Word2Vec

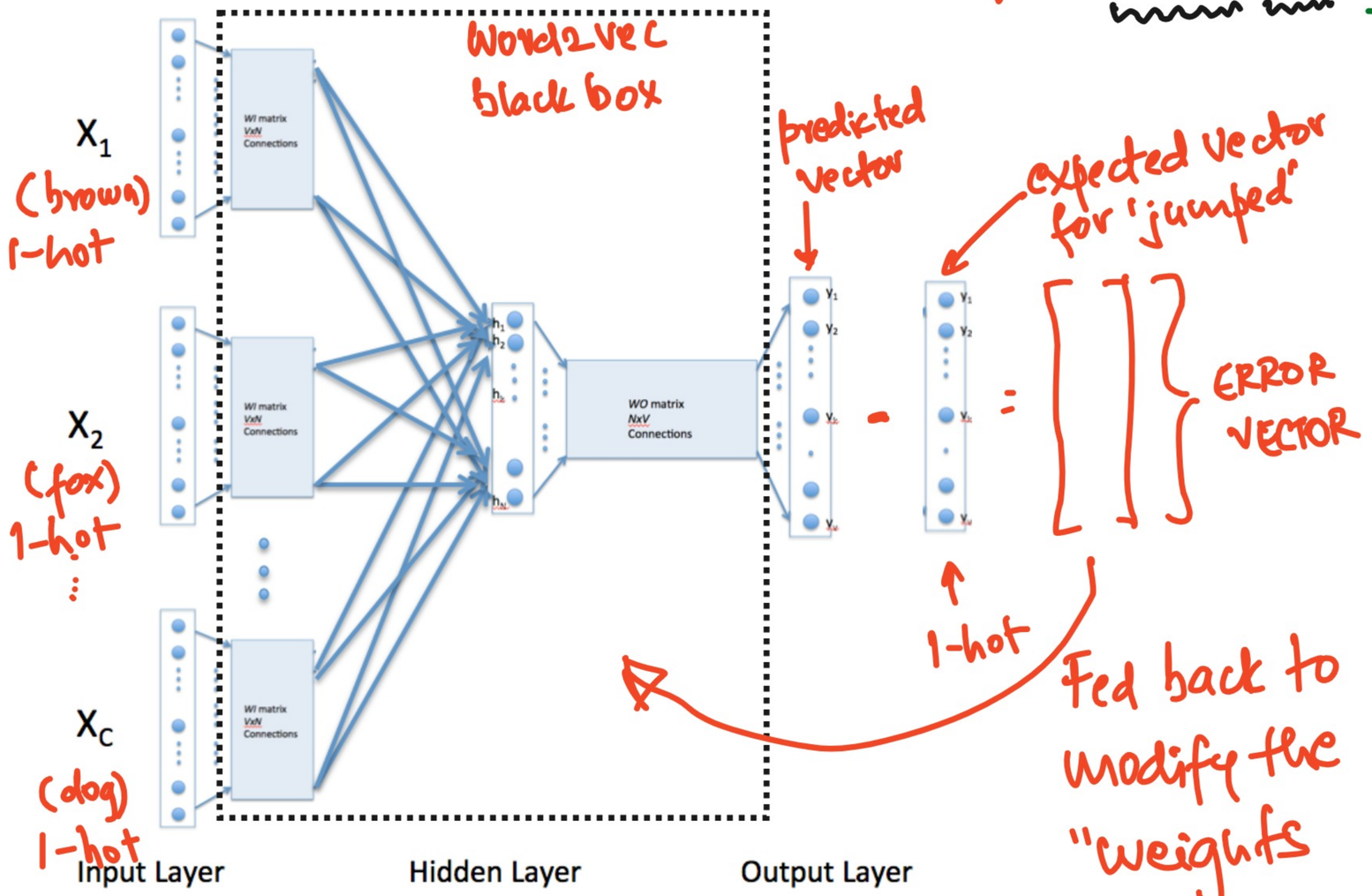
- **Captures semantic and syntactic relationships**:
  - "king" - "man" + "woman" ≈ "queen"
- **Efficient**: Can be trained on billions of words in a few hours.
- **Unsupervised**: Requires no labels.

Applications

- Text classification
- Similarity/distance computation
- Clustering documents
- Input to other models like RNNs, CNNs, Transformers
- Analogies and word reasoning

Creation of word2vec Embeddings

"The quick brown fox jumped ~~over the~~ lazy dog"



Fed back to modify the "weights

Process repeats till vectors don't change much / no of iterations are reached / training data is consumed -

https://stackoverflow.com/questions/42603417/doubts-regarding-word2vec-continuous-bag-of-words-cbow

How are word embeddings used: Example: Training of word prediction models

- Assume you have a large corpus of text, and you would like to use the corpus to train an ML model to predict 'the next word' based on the previous **three** words
- Assume the following statement is part of such a corpus: **The quick brown fox jumped over the lazy dog**
- We have to first create **training data** which we can use to train the model. Lets see how to do that.
- We need to create a series of training data entries of the form:
    - {w1, w2, w3} -> y
    - Here w1, w2, w3 are the predictors and **y** is the predicted word
- Lets create such training entries using the above statement:
    - {the, quick, brown} -> fox
    - {quick, brown, fox} -> jumped
    - {brown, fox, jumped} -> over
    - {fox, jumped, over} -> the
    - {jumped, over, the} -> lazy
    - {over, the, lazy} -> dog
- A data set comprising hundreds and thousands of such **training entries** are created using the entire corpus and, using them, prediction models can be trained.
- **Note: word2vec embeddings of the words are used during the training process. Recall that embeddings are vectors of numbers.**
- Once a prediction model (Neural Network) is thus trained and deployed, given the word2vec embedding of three prior words it has the capability to predict the embedding of the 4th (following) word.