Ans1: Here is the word-context matrix for "eyes", "here", and "valley" using a three-word window:

	The	eyes	are	not	here	There	are	no	eyes	here	In	this	valley	of	dying	stars	In	this	hollow	valley	This	broken	jaw	of	our	lost	kingdoms
eyes	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
here	0	0	0	0	0	1	1	0	1	0	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0	
valley	0	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	2	2	1	1	1	1	1	1	1	

To calculate the similarity between "eyes" and "valley," we can use the cosine similarity measure. First, we need to represent the two words as vectors, which can be done by extracting their rows from the matrix above:

eyes: [0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
valley: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1]
Next, we can calculate the cosine similarity between the two vectors:

```
similarity = dot(eyes, valley) / (norm(eyes) * norm(valley))
           = (0*0 + 0*0 + 0*0 + 0*0 + 1*0 + 0*0 + 0*0 + 0*0 + 0*0 + 1*0 + 0*1 + 0*1 + 0*0 + 0*1
              0*1 + 0*1 + 0*0 + 0*1 + 0*1 + 0*1 + 0*1 + 0*1 + 0*2 + 0*1 + 0*1 + 0*1 + 0*1 + 0*1 + 0*1
            / (√(0^2 + 0^2 + 0^2 + 0^2 + 1^2 + 0^2 + 0^2 + 0^2 + 0^2 + 1^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2) *
              √(0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 0^2 + 1^2 + 1^2 + 0^2 + 1^2 + 1^2 + 1^2 + 2^2 + 2^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2))

           = 0 / (sqrt(2) * sqrt(19))
           = 0
```
Therefore, the similarity between "eyes" and "valley" is 0. This suggests that these two words are not closely related in the given corpus.

Ans2: Word2Vec is a neural network-based algorithm for learning word embeddings, which are vector representations of words in a high-dimensional space. The Skip-gram model is a popular variant of Word2Vec that predicts the context words given a target word. Negative sampling is a technique used to simplify the learning task and speed up training.

Here's how the Skip-gram model with negative sampling works:

Given a corpus of text, create a vocabulary of unique words, and assign each word a unique ID.

For each word in the corpus, generate training examples of the form (target word, context word), where the target word is the center word and the context word is one of the words that appear within a fixed-size window around the target word.

For each training example (target word, context word), use the Skip-gram model to predict the probability of the context word given the target word. Specifically, the model uses a softmax function to compute the conditional probability of each context word in the vocabulary, given the target word:

P(context word | target word) = exp(v_c . v_t) / sum(exp(v_c . v_i))

where v_c and v_t are the vector representations of the context word and target word, respectively, and v_i are the vector representations of all other words in the vocabulary.

The model is trained by maximizing the log-likelihood of the observed context words given the target words. This is equivalent to minimizing the negative log-likelihood, which is known as the cross-entropy loss. However, computing the softmax function over the entire vocabulary can be computationally expensive, especially for large vocabularies.

To speed up training, the Skip-gram model uses negative sampling, which involves randomly selecting a small number (typically between 5 and 20) of "negative" context words for each target word. Negative context words are chosen randomly from the vocabulary, but with a probability that is proportional to their frequency in the corpus. The objective of the model is to maximize the probability of the observed context word and minimize the probability of the negative context words.

To do this, the model uses a modified version of the softmax function, called the negative sampling loss function, which is defined as follows:

J = -log(sigma(v_c . v_t)) - sum_{i=1}^k log(sigma(-v_i . v_t))

where sigma(x) = 1 / (1 + exp(-x)), k is the number of negative context words, and v_i are the vector representations of the negative context words.

During training, the parameters of the model (i.e., the word vectors) are updated using stochastic gradient descent to minimize the negative sampling loss function.
In summary, the Skip-gram model with negative sampling is a variant of the Word2Vec algorithm that predicts the context words given a target word. It uses a modified softmax function to compute the conditional probability of the context words and a negative sampling loss function to speed up training by selecting a small number of negative context words. The model is trained using stochastic gradient descent to minimize the negative sampling loss function and learn high-quality word embeddings.

Ans3: Named Entity Recognition (NER) is a natural language processing (NLP) task that involves identifying and classifying named entities in text into predefined categories such as people, organizations, locations, and others. NER helps to extract structured information from unstructured text and is useful in a wide range of applications, including information retrieval, question answering, text summarization, and machine translation.

Examples of named entities include:

Person: John Smith, Barack Obama, Mary Jane
Organization: Microsoft, Google, United Nations
Location: New York, Paris, Tokyo
Date and Time: January 1st, 2022, 2:00 PM
Monetary Value: $10,000, 1.5 million euros
Percentage: 50%, 75.5%
Product: iPhone, Tesla Model S, Coca-Cola
For example, consider the following sentence: "John Smith works as a software engineer at Google in New York."

A named entity recognition system would identify the following named entities:

Person: John Smith
Organization: Google
Location: New York
NER can also identify relations between named entities, such as "John Smith works at Google." This information can be used for knowledge extraction and various applications in natural language processing.

Ans4. Word vectors, also known as word embeddings, are numerical representations of words in a high-dimensional space. They are created using machine learning algorithms that are trained on large collections of text to learn the relationships between words.

The basic idea behind word vectors is to represent each word as a vector of real numbers, where the values of the vector capture the meaning and context of the word in the text. Word vectors are typically high-dimensional, with hundreds or thousands of dimensions, and are learned in an unsupervised manner using techniques such as Word2Vec and GloVe.

Here is a simple diagram illustrating the concept of word vectors:

Word Vector Diagram

In this diagram, we see a 2D space where each word is represented as a vector. Each word is mapped to a point in this space, and similar words are placed close together. For example, "cat" and "dog" are similar words, so they are placed close together. "car" and "bus" are also similar words, but they are placed in a different direction.

The cosine similarity between two word vectors is a measure of their similarity or distance in the high-dimensional space. It is calculated by taking the cosine of the angle between the two vectors. The cosine similarity between two vectors ranges from -1 to 1, where -1 means the vectors are completely dissimilar, 0 means they are orthogonal or independent, and 1 means they are identical.

The cosine similarity between two word vectors x and y can be calculated as follows:

cosine similarity(x, y) = (x . y) / (||x|| * ||y||)

where x . y is the dot product of x and y, and ||x|| and ||y|| are the magnitudes or lengths of the vectors x and y, respectively.

For example, consider the following word vectors in a 2D space:

"cat" = (1, 2)
"dog" = (2, 3)
"car" = (3, 1)
"bus" = (1, 3)

The cosine similarity between "cat" and "dog" can be calculated as follows:

cosine similarity("cat", "dog") = ((1 * 2) + (2 * 3)) / (√(1^2 + 2^2) * √(2^2 + 3^2)) = 0.98

The high cosine similarity value indicates that "cat" and "dog" are similar words in the 2D space. Similarly, the cosine similarity between "cat" and "car" is -0.56, indicating that they are dissimilar words.

Word vectors have become an important tool in natural language processing (NLP) because they enable algorithms to capture the meaning and context of words in a more efficient and accurate way than traditional methods. Some benefits of using word vectors include:

Improved accuracy in NLP tasks: Word vectors can capture the semantic and syntactic relationships between words, which makes them useful in a wide range of NLP tasks such as machine translation, sentiment analysis, and named entity recognition.

Reduced dimensionality: Word vectors can represent words in a high-dimensional space, but they can also be used to reduce the dimensionality of text data. This is particularly useful when working with large datasets, where traditional methods can become computationally expensive.

Transfer learning: Word vectors can be trained on large datasets and then used as a starting point for other NLP tasks. This approach, known as transfer learning, can save time and computational resources when working with smaller datasets.

There are many techniques for generating word vectors, including Word2Vec, GloVe, and FastText. These techniques are based on neural networks and use different methods for learning word representations.

In practice, word vectors are often used as inputs to other machine learning algorithms, such as neural networks or support vector machines, in order to perform various NLP tasks. By using word vectors as inputs, these algorithms can capture the meaning and context of words in a more efficient and accurate way, which can lead to improved performance on NLP tasks.

Ans5: The term-document matrix is:

Document 1		Document 2		Document 3	Document 4
digital	1	0	7	13	
computer	114	80	62	89	
information	36	58	1	4	
data	20	15	2	3	

(i) Cosine Similarity:

To calculate the cosine similarity between "good" and "fool", we first need to represent these words as vectors using the term-document matrix.

Since neither "good" nor "fool" appears in the corpus, their vectors will be zero vectors. Therefore, the cosine similarity between "good" and "fool" is 0.

(ii) PPMI (Positive Pointwise Mutual Information):

PPMI measures the association between two words based on their co-occurrence in a corpus. It is calculated as:

PPMI(w1, w2) = max(log(P(w1, w2) / (P(w1) * P(w2))), 0)

where P(w1, w2) is the probability of co-occurrence of w1 and w2, and P(w1) and P(w2) are the probabilities of occurrence of w1 and w2, respectively.

To calculate the PPMI between "good" and "fool", we first need to calculate the probabilities of their occurrence and co-occurrence in the corpus. Since neither "good" nor "fool" appears in the corpus, their probabilities of occurrence are zero. Therefore, the PPMI between "good" and "fool" is also zero.

Note: If we use add 2 smoothing to the term-document matrix to avoid zero probabilities, we would get non-zero values for the probabilities of occurrence and co-occurrence of "good" and "fool". However, since neither "good" nor "fool" co-occur with any of the words in the corpus, their PPMI would still be zero.

Ans 6: The Logistic Regression based skip-gram model is a neural network language model that uses logistic regression to predict the context words given a target word. It is a modification of the original skip-gram model, which uses softmax regression for the same purpose.

The basic architecture of the Logistic Regression based skip-gram model consists of two layers: the input layer and the output layer. The input layer represents the target word, which is transformed into a vector representation using an embedding matrix. The output layer represents the context words, which are also transformed into vector representations using the same embedding matrix.

The goal of the model is to learn the probability distribution of the context words given the target word. To achieve this, the model uses logistic regression to estimate the conditional probability of each context word given the target word.

The training process involves minimizing the negative log-likelihood of the observed context words given the target word. This is done by adjusting the weights of the embedding matrix using backpropagation.

The following diagram illustrates the architecture of the Logistic Regression based skip-gram model:

```
Input (Target Word)      Output (Context Words)
           |                      |
           v                      v
+------------------+   +-------------------+
|   Embedding      |   |   Embedding       |
|   Matrix         |   |   Matrix          |
+------------------+   +-------------------+
           |                      |
           v                      v
   Hidden Layer            Output Layer
           |                      |
           v                      v
     +----------+         +-----------+
     | Logistic |          | Logistic  |
     | Regression|        | Regression|
     +----------+         +-----------+
           |                      |
           v                      v
Predicted Context  Observed Context
           |                          |
           v                          v
+--------------------+          +-----------------------+
 |Negative Log-Likelihood|        | Gradient of Embedding |
+--------------------+          +-----------------------+
```