Ввод [1]:
```
import torch
import numpy as np
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import torch.utils.data as Data

dtype = torch.FloatTensor
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

1.Data Preparation

Ввод [2]:
```
# Define input sentences
sentences = [
    "apple orange banana mango grape lemon lime peach cherry strawberry watermelon",
    "cat dog elephant tiger lion giraffe zebra monkey panda kangaroo",
    "car bicycle motorcycle airplane train bus boat helicopter scooter skateboard",
    "computer laptop tablet smartphone printer keyboard mouse monitor",
    "ocean mountain desert forest river valley canyon lake waterfall",
    "python java javascript c++ ruby swift kotlin html css",
]
# Tokenize sentences
word_sequence = " ".join(sentences).split() # ['jack', 'like', 'dog', 'jack', 'like', 'cat', 'animal',...]
vocab = list(set(word_sequence)) # build words vocabulary
word2idx = {w: i for i, w in enumerate(vocab)} # {'jack':0, 'like':1,...}
#Tokenizes the sentences into a sequence of words, builds a vocabulary (vocab), and creates a mapping from words to
```

Ввод [3]:
```
print("Vocabulary:", vocab)
print("Word to Index Mapping:", word2idx)
```

```
Vocabulary: ['monkey', 'bicycle', 'python', 'kotlin', 'lion', 'canyon', 'zebra', 'river', 'tablet', 'mouse', 'deser
t', 'waterfall', 'ocean', 'elephant', 'watermelon', 'motorcycle', 'c++', 'cat', 'banana', 'mango', 'lake', 'panda',
'cherry', 'orange', 'helicopter', 'kangaroo', 'forest', 'keyboard', 'swift', 'valley', 'peach', 'strawberry', 'moun
tain', 'grape', 'printer', 'skateboard', 'giraffe', 'airplane', 'train', 'css', 'bus', 'scooter', 'computer', 'ca
r', 'laptop', 'apple', 'boat', 'tiger', 'java', 'javascript', 'lemon', 'html', 'ruby', 'lime', 'smartphone', 'dog',
'monitor']
Word to Index Mapping: {'monkey': 0, 'bicycle': 1, 'python': 2, 'kotlin': 3, 'lion': 4, 'canyon': 5, 'zebra': 6, 'r
iver': 7, 'tablet': 8, 'mouse': 9, 'desert': 10, 'waterfall': 11, 'ocean': 12, 'elephant': 13, 'watermelon': 14, 'm
otorcycle': 15, 'c++': 16, 'cat': 17, 'banana': 18, 'mango': 19, 'lake': 20, 'panda': 21, 'cherry': 22, 'orange': 2
3, 'helicopter': 24, 'kangaroo': 25, 'forest': 26, 'keyboard': 27, 'swift': 28, 'valley': 29, 'peach': 30, 'strawbe
rry': 31, 'mountain': 32, 'grape': 33, 'printer': 34, 'skateboard': 35, 'giraffe': 36, 'airplane': 37, 'train': 38,
'css': 39, 'bus': 40, 'scooter': 41, 'computer': 42, 'car': 43, 'laptop': 44, 'apple': 45, 'boat': 46, 'tiger': 47,
'java': 48, 'javascript': 49, 'lemon': 50, 'html': 51, 'ruby': 52, 'lime': 53, 'smartphone': 54, 'dog': 55, 'monito
r': 56}
```

Ввод [4]:
```
# Word2Vec Parameters
batch_size = 8
embedding_size = 2  # 2 dim vector represent one word
C = 2 # window size
voc_size = len(vocab)
#Defines parameters for the Word2Vec model, such as batch size, embedding size, window size (C), and vocabulary size
```

2.Skip-gram Generation

Ввод [5]:
```
 1.
kip_grams = []
or idx in range(C, len(word_sequence) - C):
 center = word2idx[word_sequence[idx]] # center word
 context_idx = list(range(idx - C, idx)) + list(range(idx + 1, idx + C + 1)) # context word idx
 context = [word2idx[word_sequence[i]] for i in context_idx]
 for w in context:
   skip_grams.append([center, w])

rint(skip_grams)
Generates skip-grams from the input sentences. Skip-grams are pairs of a center word and a context word within a cert
 2.
 Prepare Data for Training
Converts skip-grams into input and output data suitable for training
ef make_data(skip_grams):
 input_data = []
 output_data = []
 for i in range(len(skip_grams)):
   input_data.append(np.eye(voc_size)[skip_grams[i][0]])
   output_data.append(skip_grams[i][1])
 return input_data, output_data

 3.Convert Data to PyTorch Tensors and Create DataLoader
nput_data, output_data = make_data(skip_grams)
print(input_data, output_data)
nput_data, output_data = torch.Tensor(input_data), torch.LongTensor(output_data)
ataset = Data.TensorDataset(input_data, output_data)
oader = Data.DataLoader(dataset, batch_size, True)
```

```
[[18, 45], [18, 23], [18, 19], [18, 33], [19, 23], [19, 18], [19, 33], [19, 50], [33, 18], [33, 19], [33, 50], [33,
53], [50, 19], [50, 33], [50, 53], [50, 30], [53, 33], [53, 50], [53, 30], [53, 22], [30, 50], [30, 53], [30, 22],
[30, 31], [22, 53], [22, 30], [22, 31], [22, 14], [31, 30], [31, 22], [31, 14], [31, 17], [14, 22], [14, 31], [14,
17], [14, 55], [17, 31], [17, 14], [17, 55], [17, 13], [55, 14], [55, 17], [55, 13], [55, 47], [13, 17], [13, 55],
[13, 47], [13, 4], [47, 55], [47, 13], [47, 4], [47, 36], [4, 13], [4, 47], [4, 36], [4, 6], [36, 47], [36, 4], [3
6, 6], [36, 0], [6, 4], [6, 36], [6, 0], [6, 21], [0, 36], [0, 6], [0, 21], [0, 25], [21, 6], [21, 0], [21, 25], [2
1, 43], [25, 0], [25, 21], [25, 43], [25, 1], [43, 21], [43, 25], [43, 1], [43, 15], [1, 25], [1, 43], [1, 15], [1,
37], [15, 43], [15, 1], [15, 37], [15, 38], [37, 1], [37, 15], [37, 38], [37, 40], [38, 15], [38, 37], [38, 40], [3
8, 46], [40, 37], [40, 38], [40, 46], [40, 24], [46, 38], [46, 40], [46, 24], [46, 41], [24, 40], [24, 46], [24, 4
1], [24, 35], [41, 46], [41, 24], [41, 35], [41, 42], [35, 24], [35, 41], [35, 42], [35, 44], [42, 41], [42, 35],
[42, 44], [42, 8], [44, 35], [44, 42], [44, 8], [44, 54], [8, 42], [8, 44], [8, 54], [8, 34], [54, 44], [54, 8], [5
```

Ввод [6]:
```
# Model
class Word2Vec(nn.Module):
  def __init__(self):
    super(Word2Vec, self).__init__()

    # W and V is not Traspose relationship
    self.W = nn.Parameter(torch.randn(voc_size, embedding_size).type(dtype))
    self.V = nn.Parameter(torch.randn(embedding_size, voc_size).type(dtype))

  def forward(self, X):
    # X : [batch_size, voc_size] one-hot
    # torch.mm only for 2 dim matrix, but torch.matmul can use to any dim
    hidden_layer = torch.matmul(X, self.W) # hidden_layer : [batch_size, embedding_size]
    output_layer = torch.matmul(hidden_layer, self.V) # output_layer : [batch_size, voc_size]
    return output_layer

model = Word2Vec().to(device)
criterion = nn.CrossEntropyLoss().to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)
#Defines the Word2Vec model as a neural network module. It has two weight matrices, W and V.
```

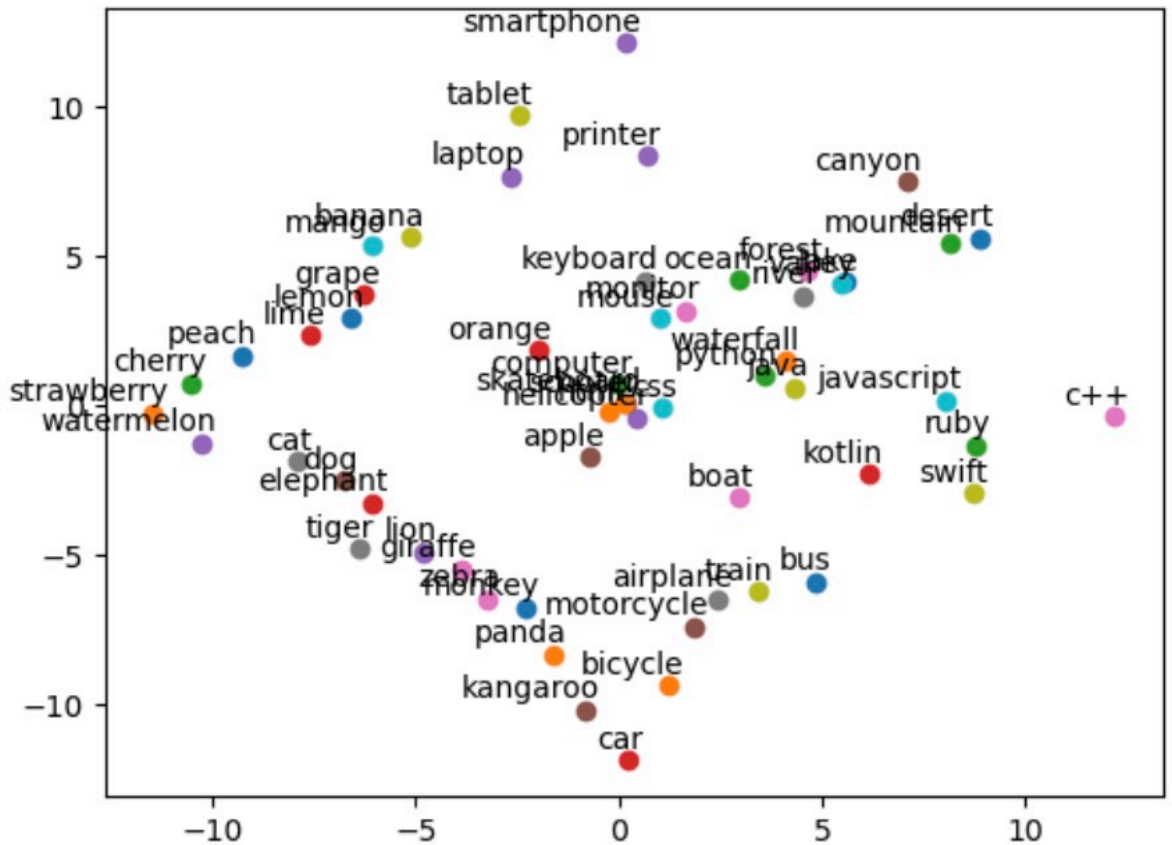
4.Training

Ввод [7]:
```
# Training
for epoch in range(2000):
  for i, (batch_x, batch_y) in enumerate(loader):
    batch_x = batch_x.to(device)
    batch_y = batch_y.to(device)
    pred = model(batch_x)
    loss = criterion(pred, batch_y)
    if (epoch + 1) % 1000 == 0:
      print(epoch + 1, i, loss.item())

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
#Conducts the training loop, iterating over epochs and batches, computing loss, and updating model parameters.
```

```
1000 0 2.4915342330932617
1000 1 2.466110944747925
1000 2 1.9673322439193726
1000 3 2.648550510406494
1000 4 2.4203991889953613
1000 5 2.203082799911499
1000 6 2.204019784927368
1000 7 2.213366746902466
1000 8 2.9652044773101807
1000 9 1.9857935905456543
1000 10 2.270608425140381
1000 11 2.649318695068359
```

Ввод [8]:
```
for i, label in enumerate(vocab):
  W, WT = model.parameters()
  x,y = float(W[i][0]), float(W[i][1])
  plt.scatter(x, y)
  plt.annotate(label, xy=(x, y), xytext=(5, 2), textcoords='offset points', ha='right', va='bottom')
plt.show()
#Plots the word vectors in a 2D space.
```



Ввод [10]:
```
#Retrieves and prints the word vector for a specific word ("jack").
word_to_lookup = "car"
word_index = word2idx[word_to_lookup]
word_vector = model.W[word_index].detach().cpu().numpy()
print(f"The word vector for '{word_to_lookup}': {word_vector}")
```

```
The word vector for 'car': [  0.25554278 -11.893168  ]
```