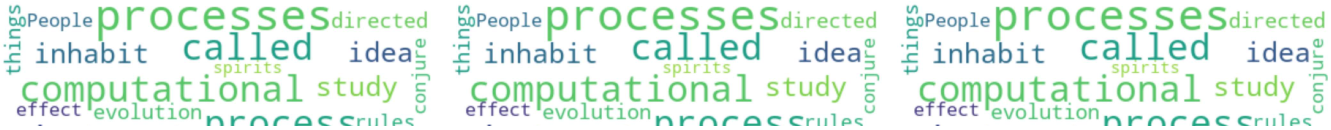
```
import re

import numpy as np
import string

import pandas as pd
import matplotlib as mpl

import matplotlib.pyplot as plt

from subprocess import check_output
from wordcloud import WordCloud, STOPWORDS


stopwords = set(STOPWORDS)
data ="""We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In effect,
we conjure the spirits of the computer with our spells."""


wordcloud = WordCloud(
                          background_color='white',
                          stopwords=stopwords,
                          max_words=200,
                          max_font_size=40,
                          random_state=42
                         ).generate(data)


fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(24, 24))
axes[0].imshow(wordcloud)
axes[0].axis('off')
axes[1].imshow(wordcloud)
axes[1].axis('off')
axes[2].imshow(wordcloud)
axes[2].axis('off')
fig.tight_layout()
```



```
# Clean Data
sentences = """We are about to study the idea of a computational process.
Computational processes are abstract beings that inhabit computers.
As they evolve, processes manipulate other abstract things called data.
The evolution of a process is directed by a pattern of rules
called a program. People create programs to direct processes. In effect,
we conjure the spirits of the computer with our spells."""


#strip it remove the space from the words
# remove special characters
sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)

# remove 1 letter words
sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()

# lower all characters
sentences = sentences.lower()


#Vocabulary
words = sentences.split()
vocab = set(words)


vocab_size = len(vocab)
embed_dim = 10
context_size = 2


# Creating the Dictionaries

word_to_ix = {word: i for i, word in enumerate(vocab)}
ix_to_word = {i: word for i, word in enumerate(vocab)}


ix_to_word
```

```
{0: 'idea',
 1: 'called',
 2: 'evolution',
 3: 'direct',
 4: 'of',
 5: 'spirits',
 6: 'rules',
 7: 'abstract',
 8: 'computational',
 9: 'directed',
 10: 'by',
     11: 'to',
     12: 'beings',
     13: 'in',
     14: 'about',
     15: 'manipulate',
     16: 'are',
     17: 'conjure',
     18: 'is',
     19: 'create',
     20: 'spells',
     21: 'programs',
     22: 'process',
     23: 'computer',
     24: 'processes',
     25: 'other',
     26: 'we',
     27: 'the',
     28: 'inhabit',
     29: 'data',
     30: 'pattern',
     31: 'study',
     32: 'computers',
     33: 'as',
     34: 'effect',
     35: 'things',
     36: 'people',
     37: 'evolve',
     38: 'with',
     39: 'program',
     40: 'our',
     41: 'they',
     42: 'that'}
```

```
# DataBags

# data - [(context), target]

data = []
for i in range(2, len(words) - 2):
    context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
    target = words[i]
    data.append((context, target))
print(data[:5])
```

```
[(['we', 'are', 'to', 'study'], 'about'), (['are', 'about', 'study', 'the'], 'to'), (['
```

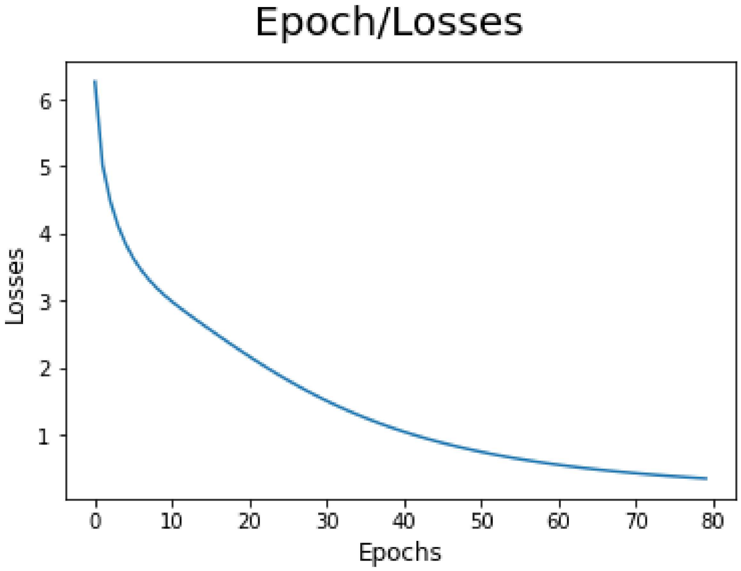
```
# Embeddings

embeddings =  np.random.random_sample((
    vocab_size, embed_dim))
```

```
embeddings
```

```
array([[2.31585517e-01, 2.53840137e-01, 1.39722670e-01, 6.35410322e-01,
        4.16835143e-01, 2.49932199e-01, 8.69380352e-01, 4.29221390e-01,
        1.52263872e-01, 7.90368690e-01],
       [5.23198900e-01, 5.33213278e-01, 3.85518778e-01, 3.98890445e-01,
        4.39098508e-02, 3.68219335e-01, 6.63102004e-01, 1.46729194e-02,
        9.66478639e-01, 2.57878910e-01],
       [4.84209824e-02, 5.09776064e-01, 5.28381281e-02, 3.45082533e-01,
        2.85914682e-01, 9.55004605e-01, 8.85108662e-01, 9.79589334e-02,
        5.33683649e-01, 5.71478044e-01],
       [4.59194104e-01, 1.14977681e-02, 9.08019622e-01, 1.73311557e-02,
        2.98292915e-04, 6.65112421e-01, 3.17269800e-02, 9.98601570e-01,
        5.69750734e-01, 4.98668648e-01],
       [7.99936288e-01, 1.51146353e-02, 6.65790710e-01, 8.19893069e-01,
        6.01475150e-02, 7.35923825e-01, 4.50307247e-02, 9.33770416e-01,
        4.55246775e-01, 5.82141703e-01],
       [4.81339378e-01, 7.20870920e-01, 6.38114866e-01, 8.52910211e-01,
        6.46868545e-01, 6.49590288e-01, 9.31698868e-01, 7.78070978e-01,
        9.41752196e-01, 5.68255292e-01],
       [4.22459812e-01, 2.45997792e-01, 2.50041219e-01, 4.58011758e-01,
        6.15540441e-01, 4.81994870e-01, 5.02969280e-01, 4.28309420e-01,
        2.49224107e-01, 8.06074678e-01],
       [8.53731405e-01, 2.20982780e-02, 6.86212514e-01, 4.27997493e-01,
        8.14154410e-01, 8.04049336e-01, 3.00286602e-01, 7.63905955e-01,
        2.52801476e-02, 9.07413087e-01],
       [9.21373877e-02, 2.14486729e-02, 9.12999162e-01, 4.44189200e-01,
        9.70358771e-01, 5.26214040e-01, 3.41667369e-01, 4.37982164e-02,
        4.68773730e-01, 3.91210473e-01],
       [8.24020027e-01, 2.09855430e-01, 5.22081325e-03, 5.18431638e-01,
        6.75146695e-01, 3.53306947e-03, 1.07456100e-01, 2.33832219e-01,
        8.81974444e-01, 7.85013838e-01],
       [2.64800906e-01, 5.93487449e-01, 1.07412453e-01, 6.34638828e-01,
        3.36191215e-01, 1.21577484e-01, 4.64396095e-01, 6.64964792e-01,
        8.96465046e-01, 7.76376469e-01],
       [5.65660256e-01, 2.30761737e-01, 2.76756962e-01, 1.87877063e-01,
        9.91112099e-01, 4.76624544e-01, 7.74160280e-01, 6.53493510e-01,
        2.02621712e-01, 7.77894568e-01],
       [9.98787699e-02, 7.59785004e-01, 9.81000276e-01, 1.35288585e-02,
        1.99414509e-01, 6.99958939e-01, 5.75305638e-01, 1.63961816e-01,
        3.11749531e-02, 3.12517678e-01],
       [1.43922667e-01, 1.12803441e-01, 8.33763577e-01, 5.15405744e-01,
        5.72342925e-01, 3.03255582e-01, 5.15296984e-01, 7.63541655e-01,
        3.92249176e-01, 5.52624631e-01],
       [8.85121947e-01, 4.14367156e-02, 3.66530665e-01, 2.82530010e-01,
        9.06174565e-01, 7.52409655e-01, 8.44869210e-02, 5.63512862e-02,
        1.65351655e-01, 5.52875233e-02],
       [2.47109767e-01, 8.21898927e-01, 1.35482544e-01, 1.65796712e-01,
        9.37609799e-01, 2.10693611e-01, 1.34051771e-01, 1.06651417e-01,
        7.06068997e-01, 2.71505396e-01],
       [5.36709483e-01, 7.01781211e-02, 2.50756322e-01, 9.37415800e-01,
        3.60648242e-01, 1.61011326e-01, 7.11369997e-02, 8.58910154e-01,
        1.35685621e-01, 1.12612973e-02],
       [4.83594828e-02, 7.74786466e-01, 8.72729642e-01, 4.38653956e-01,
        4.82947329e-01, 7.49237082e-01, 8.88889425e-01, 4.22773400e-01,
        8.01444229e-01, 9.87611757e-01],
          [1.69305795e-01, 1.43831048e-01, 3.91816921e-01, 8.80325062e-01,
           9.41701945e-01, 8.34510538e-01, 4.01730673e-01, 8.36811308e-01,
           6.42015720e-01, 2.36029949e-01],
          [6.34715288e-01, 5.00664952e-01, 6.79470773e-01, 4.90333210e-01,
```

```
# Linear Model
def linear(m, theta):
    w = theta
    return m.dot(w)


def log_softmax(x):
    e_x = np.exp(x - np.max(x))
    return np.log(e_x / e_x.sum())
def NLLLoss(logs, targets):
    out = logs[range(len(targets)), targets]
    return -out.sum()/len(out)


def log_softmax_crossentropy_with_logits(logits,target):

    out = np.zeros_like(logits)
    out[np.arange(len(logits)),target] = 1

    softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True)
    return (- out + softmax) / logits.shape[0]
def forward(context_idxs, theta):
    m = embeddings[context_idxs].reshape(1, -1)
    n = linear(m, theta)
    o = log_softmax(n)

    return m, n, o
def backward(preds, theta, target_idxs):
    m, n, o = preds
    dlog = log_softmax_crossentropy_with_logits(n, target_idxs)
    dw = m.T.dot(dlog)
    return dw
def optimize(theta, grad, lr=0.03):
    theta -= grad * lr
    return theta


#Training

theta = np.random.uniform(-1, 1, (2 * context_size * embed_dim, vocab_size))
epoch_losses = {}

for epoch in range(80):

    losses =  []

    for context, target in data:
        context_idxs = np.array([word_to_ix[w] for w in context])
        preds = forward(context_idxs, theta)

        target_idxs = np.array([word_to_ix[target]])
        loss = NLLLoss(preds[-1], target_idxs)

        losses.append(loss)

        grad = backward(preds, theta, target_idxs)
        theta = optimize(theta, grad, lr=0.03)


    epoch_losses[epoch] = losses


ix = np.arange(0,80)


fig = plt.figure()
fig.suptitle('Epoch/Losses', fontsize=20)
plt.plot(ix,[epoch_losses[i][0] for i in ix])
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Losses', fontsize=12)
```

```
     Text(0, 0.5, 'Losses')
```



```
# predict funtion

def predict(words):
    context_idxs = np.array([word_to_ix[w] for w in words])
    preds = forward(context_idxs, theta)
    word = ix_to_word[np.argmax(preds[-1])]

    return word
```

```
# (['we', 'are', 'to', 'study'], 'about')
predict(['we', 'are', 'to', 'study'])
```

```
'about'
```

```
# more than 90% accuracy is not shown in nlp
# Accuracy

def accuracy():
    wrong = 0

    for context, target in data:
        if(predict(context) != target):
            wrong += 1

    return (1 - (wrong / len(data)))
```

```
accuracy()
```

```
1.0
```