Demo of Gibbs sampling for estimating Latent Dirichlet Allocation (LDA) topic models

Adapted/simplified from code for the following article
https://naturale0.github.io/2021/02/16/LDA-4-Gibbs-Sampling
(https://naturale0.github.io/2021/02/16/LDA-4-Gibbs-Sampling)

Original code at: https://github.com/naturale0/NLP-Do-It-Yourself/blob/cfc99291774e8f4f5b93941e680df50f8ac965de/NLP_with_PyTorch/3_document-embedding/3-1.%20latent%20dirichlet%20allocation.ipynb (https://github.com/naturale0/NLP-Do-It-Yourself/blob/cfc99291774e8f4f5b93941e680df50f8ac965de/NLP_with_PyTorch/3_document-embedding/3-1.%20latent%20dirichlet%20allocation.ipynb)

In [1]:
```
from scipy.special import psi, polygamma, gammaln
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Import and preprocess Classic3 dataset

Includes abstracts from:

- Medicine
- Aerospace
- Information and library science

In [2]:
```python
import pandas as pd
import scipy.sparse

vocab_3 = pd.read_csv('classic3/word-list.txt', header=None)
doc_word_matrix_3 = pd.read_csv('classic3/doc-word-matrix.csv', header=No

# Extract vocab
max_vocab = 1000
vocab = vocab_3[1].tolist()[:max_vocab] # 0 indexed now

# Extract documents
i = doc_word_matrix_3[0].to_numpy() - 1 # zero indexed
j = doc_word_matrix_3[1].to_numpy() - 1 # zero indexed
data = doc_word_matrix_3[2].to_numpy() # counts
docs = []
for ii, jj, dd in zip(i,j,data):
  while ii >= len(docs): # In case there are empty docs
    docs.append([])
  if jj < max_vocab:
    docs[ii] += [jj] * dd

# Shuffle and convert to numpy arrays
rng = np.random.RandomState(0)
docs = rng.permutation(np.array([
  rng.permutation(np.array(doc))
  for doc in docs if len(doc) > 0
], dtype=object))

print('Num docs:', len(docs))
print('Vocab size (dimensionality):', len(vocab))
print('')
print('Two example docs')
print(docs[:2])
print('')
print('Top 10 words')
print(vocab[:10])
```

```
Num docs: 3890
Vocab size (dimensionality): 1000

Two example docs
[array([545,  35, 858,  15, 683,   0,  29, 100, 319, 125, 697,  27, 82
3,
          0, 911, 237, 339,  49,   2, 889,   0, 858,  10, 246,  13,  9
4,
        100,  35,  13, 504,  33])
 array([267, 188, 315, 167, 154,   1,  15,  10,  43, 340, 329, 287, 24
5,
        307, 571, 340, 241, 689, 188,   5, 154,  61, 833, 214, 445, 21
4,
        920,   1, 512, 402, 123,   1,  41,   1,  11, 154,  92, 329, 19
9,
         41, 402, 529, 643, 267,   1,   5, 728, 937, 687])
]

Top 10 words
['flow', 'information', 'results', 'pressure', 'number', 'library', 'bo
undary', 'layer', 'theory', 'data']
```

Collapsed Gibbs Sampling

In [3]:
```python
def run_gibbs(docs, vocab, n_topic, n_gibbs=2000, verbose=True, random_se
    # Set up random number generator
    rng = np.random.RandomState(random_seed)
    # Get dimensions of various things
    # V is the vocabulary size (dimensionality), k is the number of topic
    # N is a list of text lengths, M is the number of documents
    V, k, N, M = len(vocab), n_topic, np.array([doc.shape[0] for doc in d
    print(f"V: {V}\nk: {k}\nN: {N[:10]}...\nM: {M}")

    # Initialize hyperparameters / regularization parameters
    alpha = 1   # one for all k
    eta = 1     # one for all V
    print(f"α: {alpha}\nη: {eta}")

    # Initialize count matrices
    n_iw = np.zeros((k, V), dtype=int)  # (C^{WT})^T Word-topic counts
    n_di = np.zeros((M, k), dtype=int)  # C^{DT} Document-topic counts
    print(f"n_iw: dim {n_iw.shape}\nn_di: dim {n_di.shape}")

    # Initialize word-topic assignment
    N_max = max(N)   # Get the longest document
    assign = np.zeros((M, N_max, n_gibbs+1), dtype=int)  # Initalize late
    print(f"assign: dim {assign.shape}")

    # Initial assignment
    for d in range(M):
        for n in range(N[d]):
            # randomly assign topic to word w_{dn}
            w_dn = docs[d][n]
            assign[d, n, 0] = rng.randint(k)

            # increment counters
            i = assign[d, n, 0]
            n_iw[i, w_dn] += 1
            n_di[d, i] += 1

    # Function to compute conditional probability
    def _conditional_prob(w_dn, d):
        prob = np.empty(k)
        for i in range(k):
            # P(z_i | d)
            _1 = (n_di[d, i] + alpha) / (n_di[d, :].sum() + k*alpha)
            # P(w_dn | z_i)
            _2 = (n_iw[i, w_dn] + eta) / (n_iw[i, :].sum() + V*eta)
            prob[i] = _1 * _2
        return prob / prob.sum() # Normalize

    if verbose:
        print("\n", "="*10, "START SAMPLER", "="*10)

    # run the sampler
    for t in range(n_gibbs):
        for d in range(M):
            for n in range(N[d]):
                w_dn = docs[d][n]

                # decrement counters
```

```
                i_t = assign[d, n, t]  # previous assignment
                n_iw[i_t, w_dn] -= 1
                n_di[d, i_t] -= 1

                # assign new topics
                prob = _conditional_prob(w_dn, d)
                i_tp1 = np.argmax(rng.multinomial(1, prob))

                # increment counter according to new assignment
                n_iw[i_tp1, w_dn] += 1
                n_di[d, i_tp1] += 1
                assign[d, n, t+1] = i_tp1

        # print out status
        print(f"Sampled {t+1}/{n_gibbs}")

    return V, k, N, M, alpha, eta, n_iw, n_di, assign
```

In [4]:
```
V, k, N, M, alpha, eta, n_iw, n_di, assign = run_gibbs(docs, vocab, n_top
```

```
V: 1000
k: 5
N: [ 31  49 109  37  41  43  29  27  75 214]...
M: 3890
α: 1
η: 1
n_iw: dim (5, 1000)
n_di: dim (3890, 5)
assign: dim (3890, 270, 11)

 ========== START SAMPLER ==========
Sampled 1/10
Sampled 2/10
Sampled 3/10
Sampled 4/10
Sampled 5/10
Sampled 6/10
Sampled 7/10
Sampled 8/10
Sampled 9/10
Sampled 10/10
```

Sanity check training results

Recover β and θ from the sample

In [5]:

```
beta = np.empty((k, V))
theta = np.empty((M, k))

for j in range(V):
    for i in range(k):
        beta[i, j] = (n_iw[i, j] + eta) / (n_iw[i, :].sum() + V*eta)

for d in range(M):
    for i in range(k):
        theta[d, i] = (n_di[d, i] + alpha) / (n_di[d, :].sum() + k*alpha)
```

Show most important words based on β

In [6]:

```
# Show most important words for each topic
def n_most_important(beta_i, n=30):
    max_values = beta_i.argsort()[-n:][::-1]
    return np.array(vocab)[max_values]

for i in range(k):
    print(f"TOPIC {i:02}: {n_most_important(beta[i], 10)}")
```

```
TOPIC 00: ['flow' 'pressure' 'boundary' 'layer' 'number' 'mach' 'shock'
'theory'
 'results' 'heat']
TOPIC 01: ['cells' 'growth' 'normal' 'increased' 'effect' 'found' 'cel
l' 'hormone'
 'increase' 'human']
TOPIC 02: ['wing' 'method' 'results' 'lift' 'theory' 'wings' 'analysis'
'methods'
 'used' 'made']
TOPIC 03: ['information' 'library' 'system' 'libraries' 'data' 'researc
h' 'use'
 'science' 'systems' 'scientific']
TOPIC 04: ['patients' 'cases' 'children' 'treatment' 'time' 'developmen
t' 'cancer'
 'case' 'study' 'changes']
```

Show topics of samples based on θ

In [7]:
```python
# Show documents and topic distribution
for ii, (doc_i, theta_i) in enumerate(zip(docs[:5], theta)):
  print(f'Doc {ii} words:"{[vocab[wi] for wi in doc_i]}"')
  print(f'Doc {ii} topics: "{theta_i}"\n')
```

```
Doc 0 words:"['attempt', 'large', 'portion', 'made', 'determination',
'flow', 'presented', 'rate', 'authors', 'blood', 'source', 'surface',
'curve', 'flow', 'needed', 'technique', 'clinical', 'ratio', 'results',
'formula', 'flow', 'portion', 'method', 'change', 'use', 'possible', 'r
ate', 'large', 'use', 'scale', 'present']"
Doc 0 topics: "[0.27777778 0.05555556 0.16666667 0.19444444 0.3055555
6]"

Doc 1 words:"['survey', 'service', 'including', 'search', 'services',
'information', 'made', 'method', 'experimental', 'users', 'indicated',
'provide', 'current', 'reported', 'sources', 'users', 'reference', 'net
work', 'service', 'library', 'services', 'based', 'dissemination', 'uni
versity', 'individual', 'university', 'sdi', 'information', 'activitie
s', 'center', 'several', 'information', 'discussed', 'information', 'sy
stem', 'services', 'computer', 'indicated', 'user', 'discussed', 'cente
r', 'previously', 'features', 'survey', 'information', 'library', 'othe
rs', 'centers', 'professional']"
Doc 1 topics: "[0.01851852 0.03703704 0.01851852 0.85185185 0.0740740
7]"

Doc 2 words:"['zero', 'theory', 'laminar', 'transverse', 'data', 'inves
tigation', 'heat', 'nose', 'angles', 'distribution', 'transfer', 'effec
ts', 'tested', 'slender', 'greater', 'layer', 'boundary', 'local', 'thi
ckness', 'theory', 'heat', 'agreement', 'effects', 'experiments', 'expe
rimental', 'shock', 'sharp', 'showed', 'shock', 'agreement', 'mach', 's
harp', 'curvature', 'zero', 'calculations', 'displacement', 'nose', 'lo
cal', 'rates', 'numbers', 'shown', 'assumption', 'hypersonic', 'cones',
'heat', 'transfer', 'tunnel', 'side', 'cone', 'cone', 'displacement',
'agreement', 'including', 'heat', 'part', 'boundary', 'layer', 'layer',
'transfer', 'included', 'conducted', 'based', 'rates', 'discussed', 'go
od', 'along', 'effects', 'heat', 'slender', 'laminar', 'experimental',
'angles', 'compared', 'theoretical', 'characteristics', 'transfer', 'go
od', 'predicted', 'slender', 'study', 'flow', 'rates', 'similarity', 'f
ound', 'cones', 'boundary', 'tests', 'heat', 'flight', 'transfer', 'hyp
ersonic', 'good', 'made', 'agreement', 'general', 'larger', 'curvatur
e', 'angles', 'layer', 'bodies', 'nose', 'transverse', 'cones', 'theor
y', 'angles', 'transfer', 'transfer', 'blunt', 'heat']"
Doc 2 topics: "[0.89473684 0.05263158 0.02631579 0.00877193 0.0175438
6]"

Doc 3 words:"['journal', 'information', 'number', 'chemical', 'technolo
gy', 'scientific', 'world', 'average', 'systems', 'technology', 'inform
ation', 'increased', 'documents', 'scientific', 'world', 'documents',
'published', 'year', 'years', 'new', 'chemical', 'technical', 'author
s', 'year', 'time', 'integral', 'last', 'publications', 'papers', 'scie
ntific', 'present', 'documents', 'literature', 'throughout', 'publishe
d', 'technical', 'chemical']"
Doc 3 topics: "[0.07142857 0.04761905 0.02380952 0.76190476 0.0952381
]"

Doc 4 words:"['observed', 'sodium', 'showed', 'different', 'function',
'hours', 'rats', 'values', 'load', 'given', 'control', 'load', 'value',
'value', 'single', 'administration', 'resulted', 'acid', 'single', 'rat
s', 'given', 'measure', 'rats', 'rats', 'water', 'control', 'test', 'in
jection', 'cent', 'given', 'renal', 'increases', 'values', 'observed',
'hours', 'acid', 'rats', 'increased', 'given', 'given', 'single']"
Doc 4 topics: "[0.06521739 0.58695652 0.26086957 0.04347826 0.0434782
```

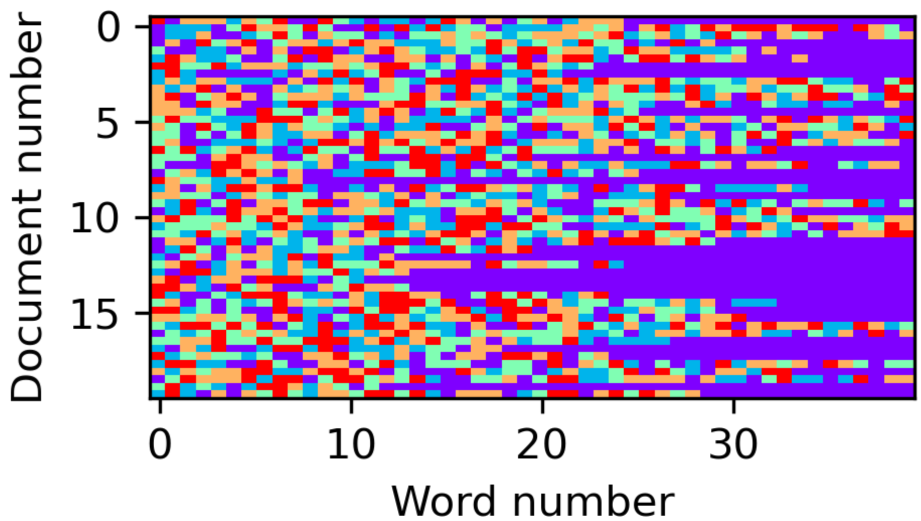
```
6]"
```

Show progression of topic assignments over time

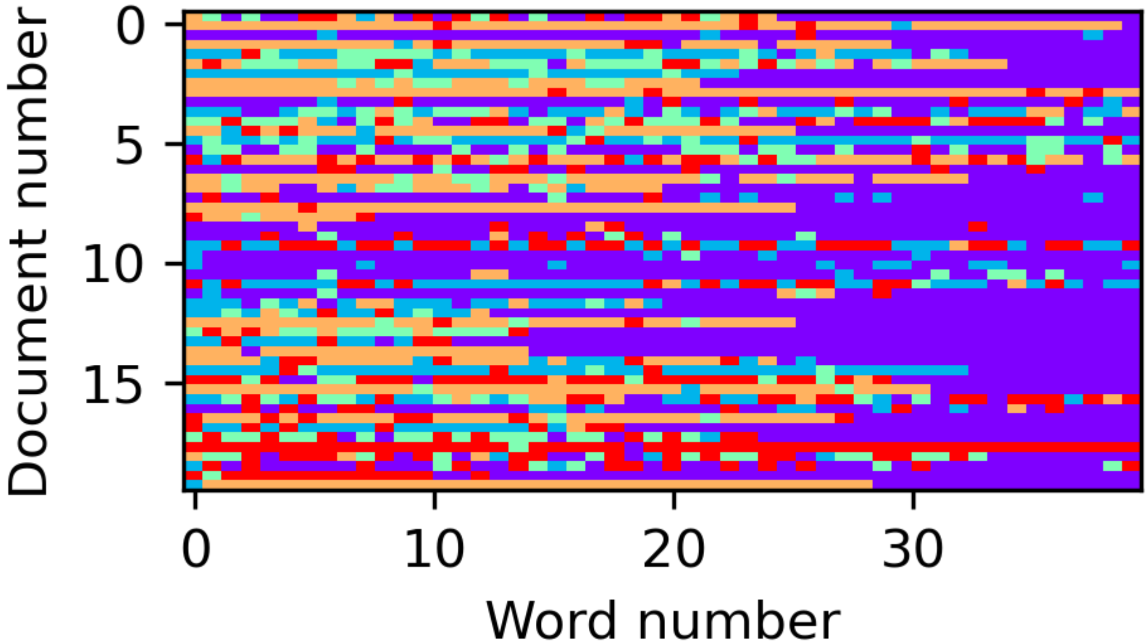
In [18]:

```
%matplotlib inline
from matplotlib.animation import FuncAnimation
# Progression of these over time
fig = plt.figure(figsize=(4,2), dpi=300)
img = plt.imshow(assign[:20,:40,0], cmap='rainbow')
plt.ylabel('Document number')
plt.xlabel('Word number')
fig.tight_layout()

def animate(i):
  img.set_array(assign[:50,:50,i])
  return img,

ani = FuncAnimation(fig, animate, frames=assign.shape[2])
#, interval=1000, blit=True, save_count=50)
from IPython.display import HTML
HTML(ani.to_jshtml())
```

Out[18]:



Once Loop Reflect


0
5
10
15
Document number
0
10
20
30
Word number