ExerciseSheet_10

July 5, 2021

1 Probabilistic Machine Learning

University of Tübingen, Summer Term 2021

1.1 Exercise Sheet 10

This sheet is **due on Tuesday 06.07.2021 at 10am**

---

2 Topic Models with LDA: Toy Example

This is the first of four exercise sheets about topic models. We will start by building a topic model on a small toy dataset and speed up the algorithm with JAX.

2.1 Part I: Toy dataset

In the "Toy_data.ipynb" file you will find a generative process for a Toy dataset. This dataset is originally from Griffiths & Steyvers PNAS 101/1 (4/2004) 5228–5235 (pdf here). If you run the notebook you will get a pickle file called "ToyData.pkl" which is required for the next parts of this exercise sheet.

2.1.1 Task 1:

Your task is two-fold.

1. Run `ExerciseSheet_10_ToyData.ipynb` notebook and generate the pickle file.
2. Give a short summary of how the toy dataset is generated. This will also make the rest of the exercise easier for you. Write about 5 bullet points with 1 sentence each. Use mathematical notation when appropriate.

Answer. 1. The K topics are created by creating K V-dim. vectors (theta) representing a distribution over words for each topic. 2. The documents are initialized by assigning each one a Dirichlet distribution (pi) over topics. 3. In each document, each word gets assigned a specific topic by sampling from the document's pi-distribution. 4. The actual words are then sampled from the theta distriubtion corresponding to the word's assigned topic. 5. In this step, the document's counter for the sampled word is incremented; this counter (vector of length V) is the final representation of the document.

[1]:
```python
import multiprocessing as mp
import pickle

#import matplotlib as mpl
import numpy as np
from matplotlib import pyplot as plt
from scipy.stats import dirichlet, multinomial
import tqdm.auto as tqdm
from einops import rearrange

# Make sure to take a look at and run ExerciseSheet_10_ToyData.ipynb first!
with open("ToyData.pkl", "rb") as f:
    DAT = pickle.load(f)

K2 = DAT["K2"]
K = DAT["K"]
D = DAT["D"]
N = DAT["N"]
V = DAT["V"]
X = DAT["X"]
PI = DAT["PI"]
THETA = DAT["THETA"]
W = DAT["W"]
ALPHA = DAT["ALPHA"]

beta = 1
alpha = ALPHA
```

2.2 Part II: Simple implementation of LDA

In this part of the exercise we will implement a first version of a topic model with the Gibbs sampling as shown in the lecture.

2.2.1 Task 2:

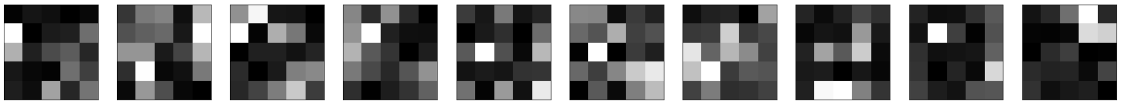
When implementing the Gibbs sampler try to 1. Use `numpy` or `scipy` whenever possible to speed up computation 2. Vectorize your computation whenever possible to save time (instead of using standard Python's loops) 3. Use easy-to-understand indices (just stick to the ones from the lecture, please) 4. Time your process, e.g. with the `tqdm` or `time` packages 5. **Required.** At each timestep, plot the current state of Θ, the topic-word proportions. After how many iteration do you see any progress? **Hint.** The first couple of iterations might look very random, so do not interupt your algorithm too early.

[2]:
```python
# initialise the counts:
C = np.zeros((D, N), dtype=int)
n_dkv = np.zeros((D, K, V))

numstep = 30
```

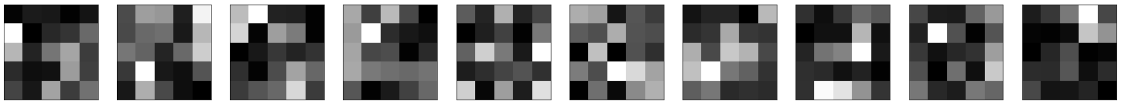
```
for step in tqdm.trange(numstep):
   ␣
 ↪#####################################################################################
    ## STEP 1: Sample topic-word proportions \Theta
   ␣
 ↪#####################################################################################

    theta = np.zeros_like(THETA)
    for k in range(K):
        theta[k] = dirichlet.rvs(beta + n_dkv.sum(0)[k])


   ␣
 ↪#####################################################################################
    ## STEP 2: Sample document-topic proportions \Pi
   ␣
 ↪#####################################################################################

    pi = np.zeros_like(PI)
    for d in range(D):
        pi[d] = dirichlet.rvs(alpha + n_dkv[d].sum(-1))


   ␣
 ↪#####################################################################################
    ## STEP 3: Sample document-topic assignments C, and update counts n_dkv
   ␣
 ↪#####################################################################################

    n_dkv = np.zeros((D, K, V))
    for d in range(D):
        for i in range(N):
            v = W[d, i]
            p = pi[d] * theta[:, v]
            p /= p.sum()
            C[d, i] = np.random.choice(K, p=p)
            n_dkv[d, C[d, i], v] += 1


   ␣
 ↪#####################################################################################
    ## Plotting
   ␣
 ↪#####################################################################################␣
 ↪
```

```
# Plotting theta is not as informative as plotting the individual "topics"
print(f"Step {step + 1}")
fig, axs = plt.subplots(1, K, figsize=(30, 3))
for k in range(K):
    axs[k].imshow(theta[k, :].reshape(K2, K2), cmap="gray")
    axs[k].set_xticks([])
    axs[k].set_yticks([])
plt.show()
```

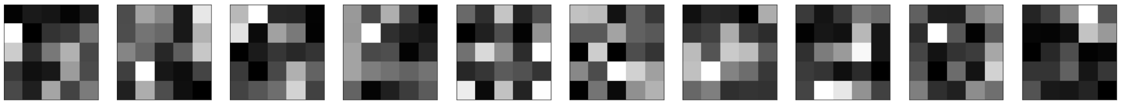
```
  0%|          | 0/30 [00:00<?, ?it/s]

Step 1
```



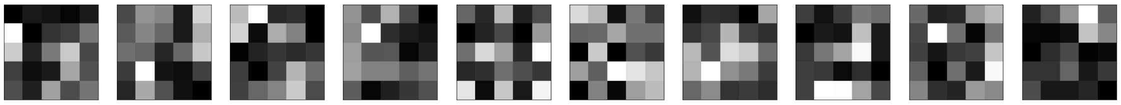
```
Step 2
```



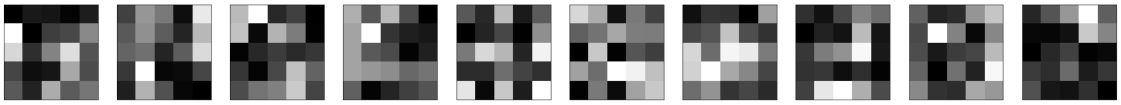
```
Step 3
```



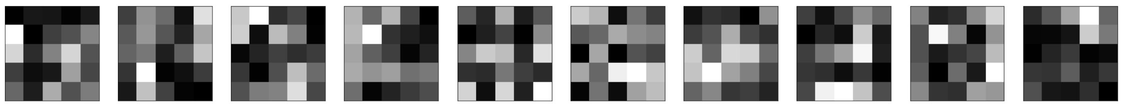
```
Step 4
```



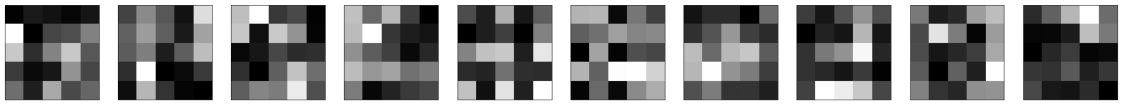
```
Step 5
```



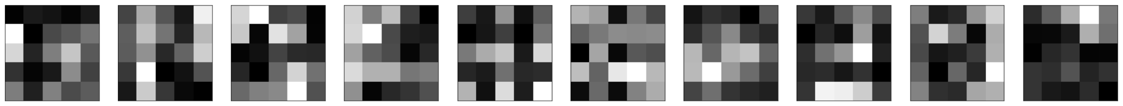
Step 6



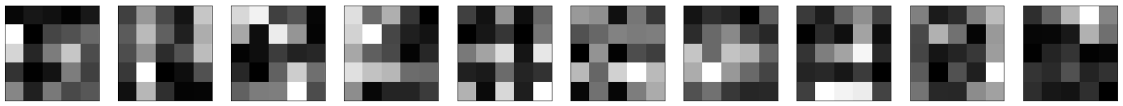
Step 7



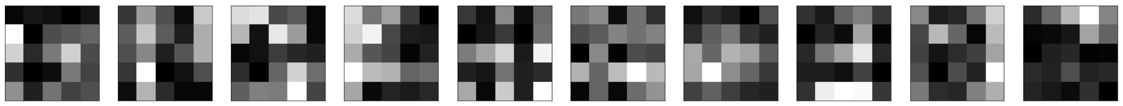
Step 8



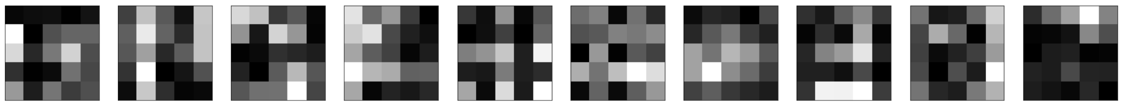
Step 9



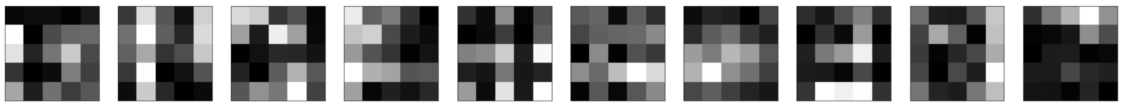
Step 10



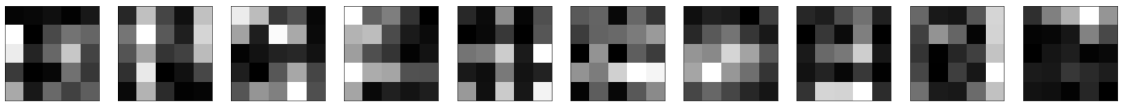
Step 11



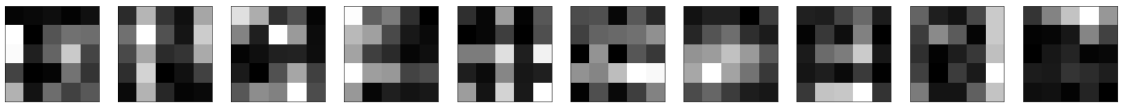
Step 12



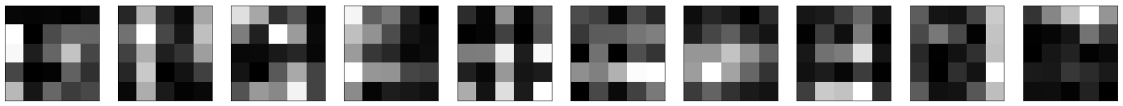
Step 13



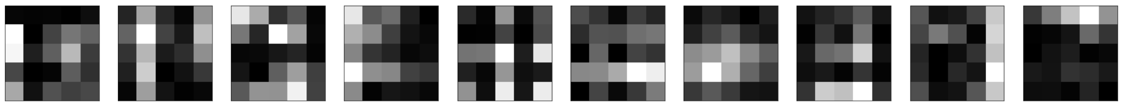
Step 14



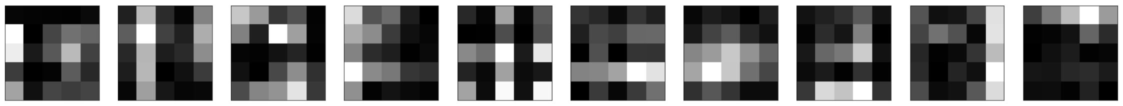
Step 15



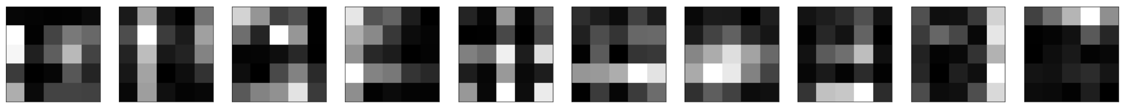
Step 16



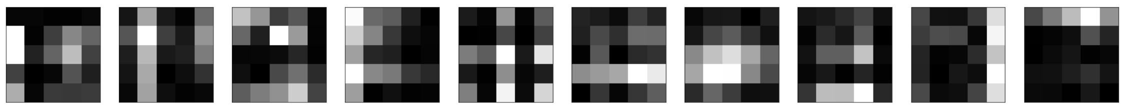
Step 17



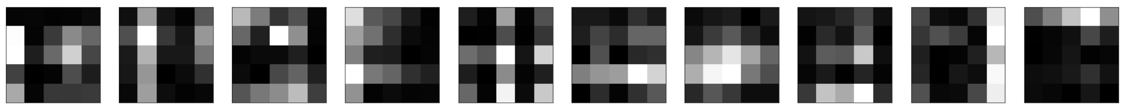
Step 18



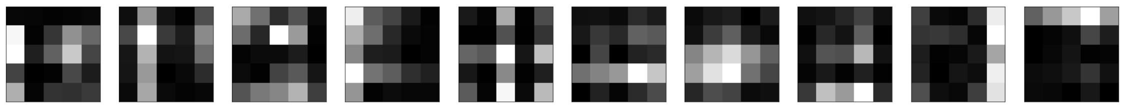
Step 19



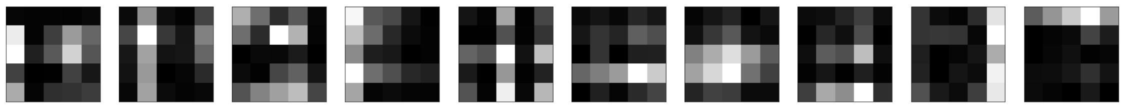
Step 20



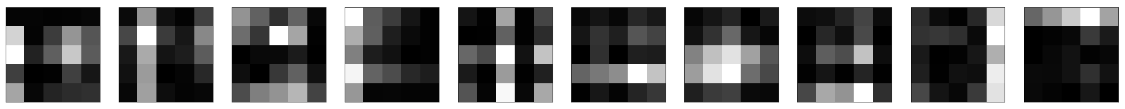
Step 21



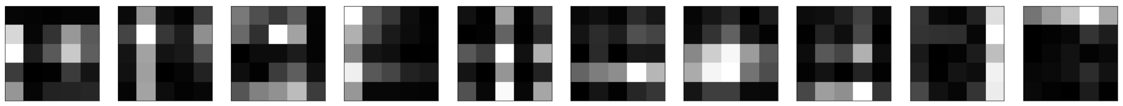
Step 22



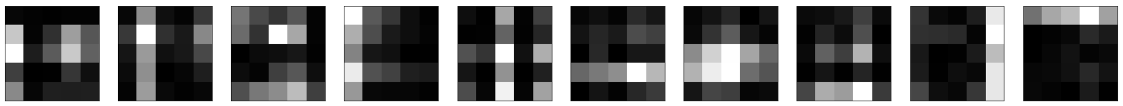
Step 23



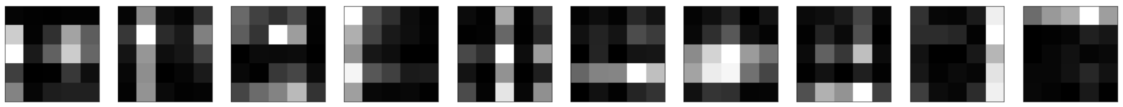
Step 24



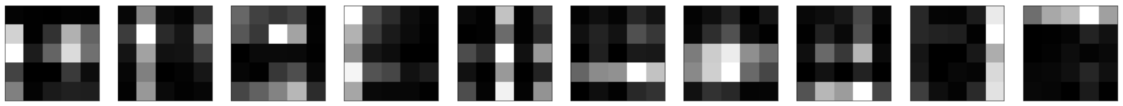
Step 25



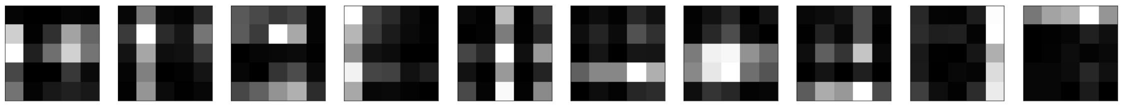
Step 26



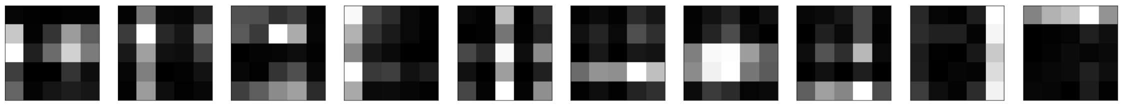
Step 27



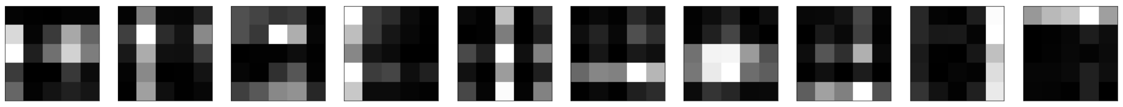
Step 28



Step 29



```
Step 30
```



2.2.2 How fast is your implementation?

For comparison, Marius' rusty computer takes: `4.40s/it`

Answer. On my computer it takes about 4.72s/it (plotting included). It is difficult to say where "progress starts", it happens very gradually. At iteration 10 it still looks rather random, however some topics are already discernable.

2.3 Part III: Make it fast with JAX

JAX is a tool that allows for fast matrix manipulations and much more. It is similar to numpy in its appearance but much more powerful. Your task is to re-implement the Gibbs sampler for the topic model above with JAX.

2.3.1 Task 3:

If you are unfamiliar with JAX, start by reading how to think JAX and JAX as accelerated NumPy.

For you implementation: 1. Consider how random number generator works in JAX 2. Try to replace remaining for-loops in NumPy. Especially look at `jax.random.dirichlet` and `jax.random.categorical`. 3. Time your implementation 4. **Required.** Plot the current state of Θ, every 10 iterations. After how many iteration do you see any progress? Run for round about 200 iterations and save the plots.

[3]:
```python
import jax
import jax.numpy as jnp
```

[5]:
```python
# initialize the counts:
n_dkv = np.zeros((D, K, V))
C = np.zeros((D, N), dtype=int)

numstep = 200
key = jax.random.PRNGKey(42)

for step in tqdm.trange(numstep):
    ␣
 ↪###########################################################################
    ## STEP 1: Sample topic-word proportions \Theta
```

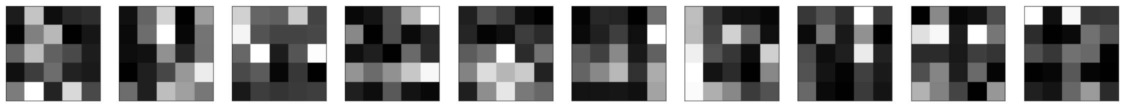
```
  ␣
↪##########################################################################################

   key, _ = jax.random.split(key)
   theta = jax.random.dirichlet(key, beta + n_dkv.sum(0))    # K x V



  ␣
↪##########################################################################################
   ## STEP 2: Sample document-topic proportions \Pi
  ␣
↪##########################################################################################

   key, _ = jax.random.split(key)
   pi = jax.random.dirichlet(key, alpha + n_dkv.sum(2))    # D x K



  ␣
↪##########################################################################################
   ## STEP 3: Sample document-topic assignments C, and update counts n_dkv
  ␣
↪##########################################################################################

   key, _ = jax.random.split(key)
   C = jax.random.categorical(key, jnp.log(theta[:, W]) + jnp.log(pi.T[...,␣
↪None]), 0)    # D x N
   C_ = rearrange(jnp.eye(K)[C], 'D N K -> D N K 1')
   W_ = rearrange(jnp.eye(V)[W], 'D N V -> D N 1 V')
   n_dkv = (C_ * W_).sum(1).astype(int)


  ␣
↪##########################################################################################
   ## Plotting
  ␣
↪##########################################################################################␣
↪

   if (step + 1) % 10 == 0:
       # Plotting theta is not as informative as plotting the individual␣
↪"topics"
       print(f"Step {step + 1}")
       fig, axs = plt.subplots(1, K, figsize=(30, 3))
       for k in range(K):
           axs[k].imshow(theta[k, :].reshape(K2, K2), cmap="gray")
           axs[k].set_xticks([])
           axs[k].set_yticks([])
```

```
        plt.show()
```

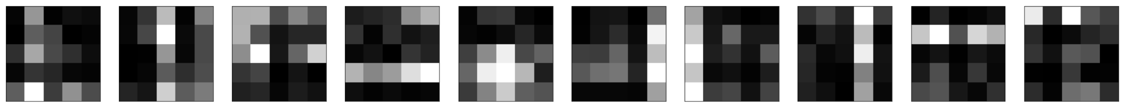
```
WARNING:absl:No GPU/TPU found, falling back to CPU. (Set TF_CPP_MIN_LOG_LEVEL=0
and rerun for more info.)
```

```
  0%|          | 0/200 [00:00<?, ?it/s]
```

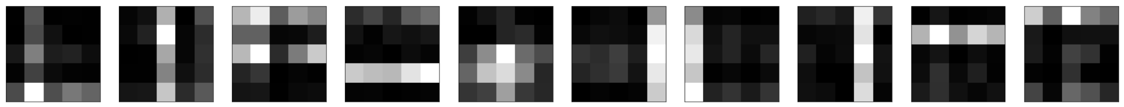
```
Step 10
```



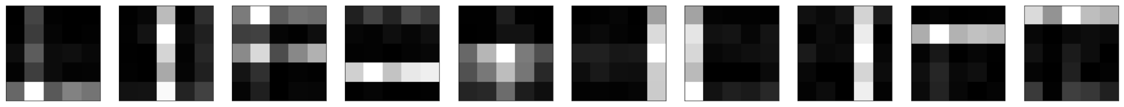
```
Step 20
```



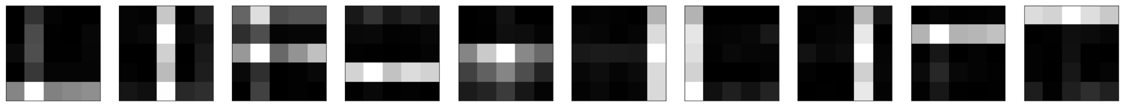
```
Step 30
```



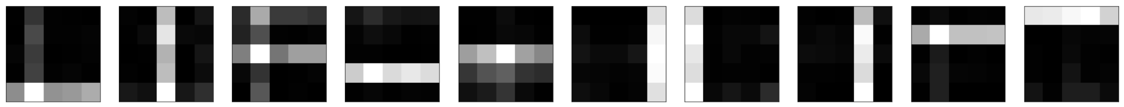
```
Step 40
```



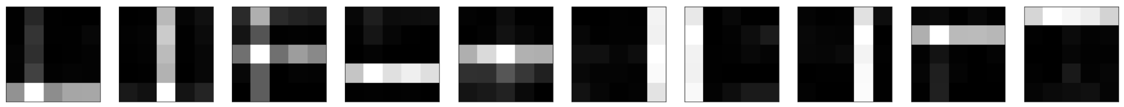
```
Step 50
```



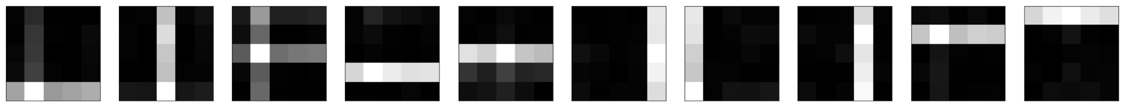
```
Step 60
```



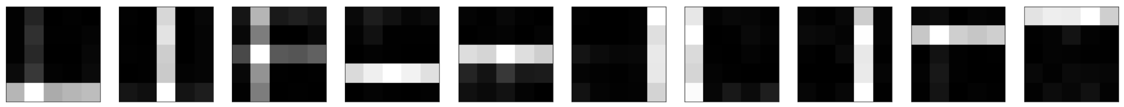
Step 70



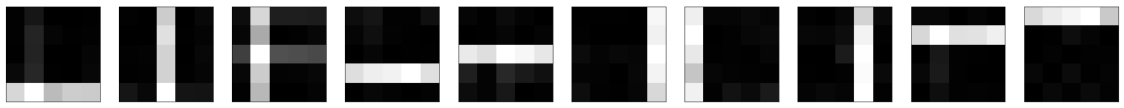
Step 80



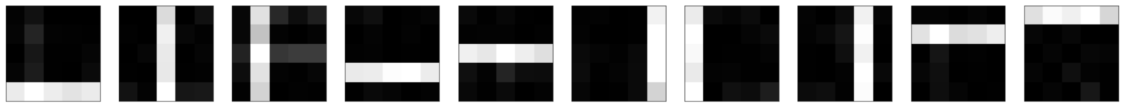
Step 90



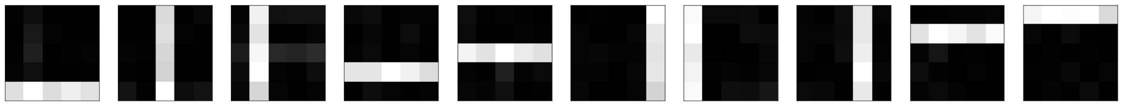
Step 100



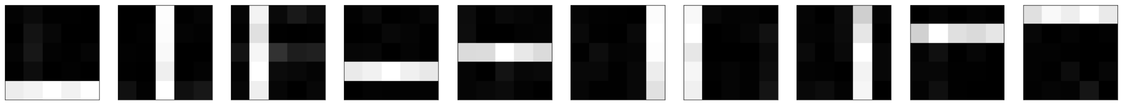
Step 110



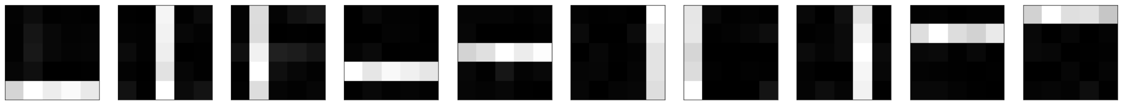
Step 120



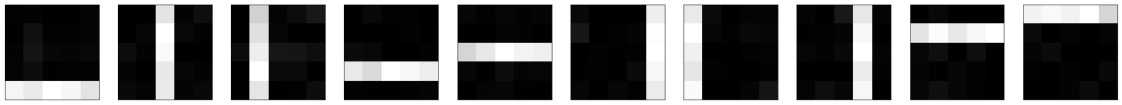
Step 130



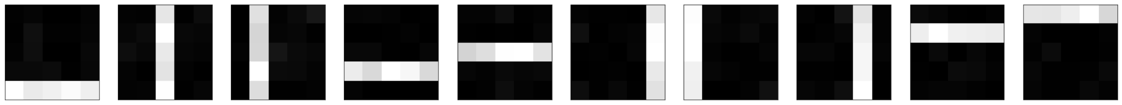
Step 140



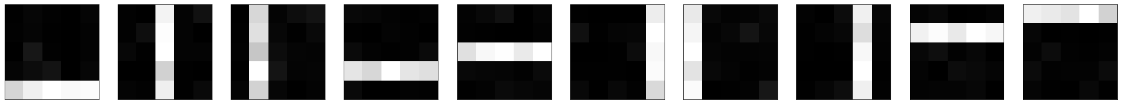
Step 150



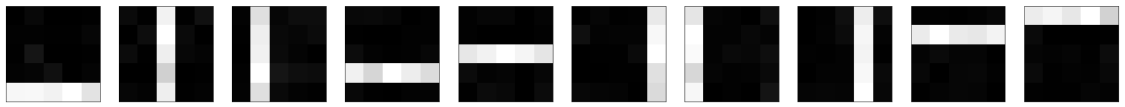
Step 160



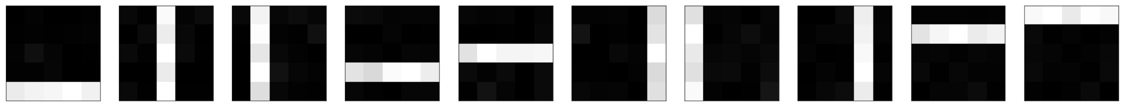
Step 170



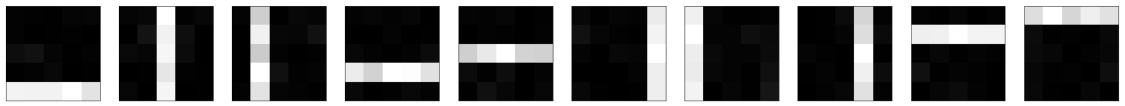
Step 180



Step 190



Step 200



2.3.2 How fast is your implementation? How does it compare to the NumPy version?

Answer. This version is much faster. The timer says 11.72it/s, however the plotting every 10 iterations noticeably slows the loop down. Again, progress happens gradually. At iteration 10 some topics are taking shape, at iteration 100 every topic is recognizable.