neural_final

January 3, 2026

1 Modèle de langue neuronal

15 décembre 2025

Adapté du tutoriel d'A. Karphathy "Makemore", deuxième partie: https://www.youtube.com/watch?v=PaCmpygFfXo

1.1 Jeu de données: les mots du code civil

[2]:
```
words = open('civil_mots.txt', 'r').read().splitlines()
chars = sorted(list(set(''.join(words))))
nb_chars = len(chars) + 1  # On ajoute 1 pour EOS
ctoi = {c:i+1 for i,c in enumerate(chars)}
ctoi['.'] = 0
print("CTOI =", ctoi)
# Dictionnaire permettant permettant de passer d'un entier à son caractère
itoc = {i:s for s,i in ctoi.items()}
print("ITOC =", itoc)
```

```
CTOI = {"'": 1, '-': 2, 'a': 3, 'b': 4, 'c': 5, 'd': 6, 'e': 7, 'f': 8, 'g': 9,
'h': 10, 'i': 11, 'j': 12, 'l': 13, 'm': 14, 'n': 15, 'o': 16, 'p': 17, 'q': 18,
'r': 19, 's': 20, 't': 21, 'u': 22, 'v': 23, 'w': 24, 'x': 25, 'y': 26, 'z': 27,
'à': 28, 'â': 29, 'ç': 30, 'è': 31, 'é': 32, 'ê': 33, 'ë': 34, 'î': 35, 'ï': 36,
'ô': 37, 'ù': 38, 'û': 39, 'œ': 40, '.': 0}
ITOC = {1: "'", 2: '-', 3: 'a', 4: 'b', 5: 'c', 6: 'd', 7: 'e', 8: 'f', 9: 'g',
10: 'h', 11: 'i', 12: 'j', 13: 'l', 14: 'm', 15: 'n', 16: 'o', 17: 'p', 18: 'q',
19: 'r', 20: 's', 21: 't', 22: 'u', 23: 'v', 24: 'w', 25: 'x', 26: 'y', 27: 'z',
28: 'à', 29: 'â', 30: 'ç', 31: 'è', 32: 'é', 33: 'ê', 34: 'ë', 35: 'î', 36: 'ï',
37: 'ô', 38: 'ù', 39: 'û', 40: 'œ', 0: '.'}
```

1.2 Approche par réseau de neurones reproduisant l'approche par comptage

1.2.1 Représentation des mots avec des vecteurs "one-hot": exemple avec un seul mot

[3]:
```
import torch

# Création d'un jeu d'entrainement de bigrams (x,y)
xs, ys = [], []
```

```
for w in [words[40]]:
  chs = ['.'] + list(w) + ['.']
  for ch1, ch2 in zip(chs, chs[1:]):
    ix1 = ctoi[ch1]
    ix2 = ctoi[ch2]
    print(ch1, ch2, '->', ix1, ix2)
    xs.append(ix1)
    ys.append(ix2)

xs = torch.tensor(xs)
ys = torch.tensor(ys)
print(words[40])
tensor_dims = len(words[40]) + 1
print("tensor_dims =", tensor_dims)
```

```
. a -> 0 3
a c -> 3 5
c c -> 5 5
c e -> 5 7
e p -> 7 17
p t -> 17 21
t é -> 21 32
é e -> 32 7
e . -> 7 0
acceptée
tensor_dims = 9
```

[4]:
```
xs
```

[4]:
```
tensor([ 0,  3,  5,  5,  7, 17, 21, 32,  7])
```

[5]:
```
ys
```

[5]:
```
tensor([ 3,  5,  5,  7, 17, 21, 32,  7,  0])
```

[6]:
```
# Représentation de chaque caractère par un vecteur one-hot
# seul une composante est à 1.0, correspondant à l'indice du numéro du caractère
import torch.nn.functional as F
xenc = F.one_hot(xs, num_classes=nb_chars).float()
xenc
```

[6]:
```
tensor([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0.],
        [0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0.],
```

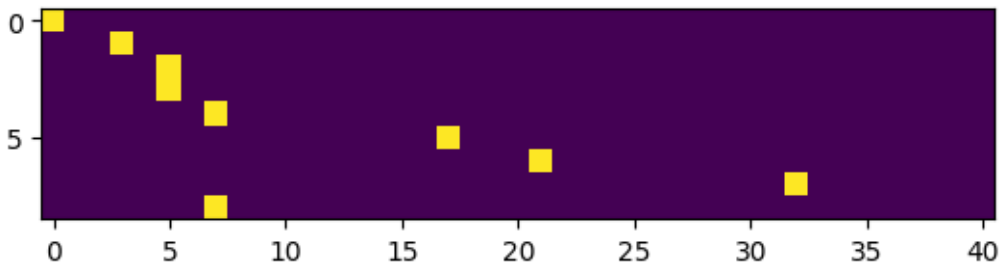
```
[0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
 0., 0., 0., 0., 0.],
[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
 0., 0., 0., 0., 0.]])
```

[7]:
```
# La première dimension est la dimension du tenseur exemple
xenc.shape
```

[7]:
```
torch.Size([9, 41])
```

[8]:
```
import matplotlib.pyplot as plt
%matplotlib inline
plt.imshow(xenc)
```

[8]:
```
<matplotlib.image.AxesImage at 0x10f450830>
```



[9]:
```
# Pour notre réseau, on va utiliser une matrice W des valeurs normales␣
 ↪aléatoires comme
# point de départ
```

```
W = torch.randn((nb_chars, tensor_dims))  # Quand on aura tous les mots, on␣
 ↪utilisera nb_chars x nb_chars
W
```

[9]:
```
tensor([[ 9.1839e-01, -1.1502e+00, -8.3629e-01,  1.5226e+00,  8.4343e-01,
          3.7099e-01,  1.8180e+00, -6.9873e-01,  1.1443e+00],
        [-8.5412e-01,  9.1880e-01,  1.6299e+00, -9.2986e-02,  4.3231e-01,
          3.4169e-01,  6.3554e-01, -2.3884e+00, -1.6909e-01],
        [ 1.1750e+00,  9.7545e-01,  6.1806e-01,  2.2869e-01, -1.2293e+00,
          8.1735e-01, -9.5131e-01,  1.2416e+00,  5.6462e-01],
        [ 2.6404e-01, -1.0920e-01,  5.7622e-01,  1.4399e-01,  7.6392e-01,
          3.0535e-01, -7.3563e-01, -9.6206e-01,  3.5338e-01],
        [ 1.5070e+00,  1.4913e+00,  2.9527e-01, -4.1975e-01, -1.4888e+00,
          6.9776e-01,  8.9753e-01,  1.0222e+00, -1.4788e+00],
        [ 1.2613e-01,  8.5963e-01, -7.1895e-01, -2.6217e+00,  4.3825e-01,
          2.3034e+00, -1.7862e+00,  3.5024e-01,  9.5713e-01],
        [-3.4914e-01,  1.1381e+00, -8.4526e-01,  3.8407e-01, -2.9689e-02,
         -8.9423e-01,  9.8505e-01,  9.0442e-01, -5.1849e-01],
        [ 6.5824e-01, -7.6243e-01, -8.5083e-02, -1.5097e+00,  3.3365e-01,
         -4.7317e-02, -5.2482e-01,  1.1375e+00, -3.8334e-01],
        [-4.3876e-02,  9.8445e-01,  1.6056e-01, -4.3068e-02, -4.3082e-01,
         -2.1318e-01, -4.0735e-01, -8.3677e-01,  3.9790e-01],
        [-1.7768e+00, -9.3988e-01,  1.0934e+00,  8.8543e-01, -1.2180e+00,
         -7.6091e-01,  5.6308e-01, -6.6491e-01, -1.1863e+00],
        [-1.4602e+00,  4.4132e-01, -3.5278e-01, -1.4347e+00, -1.5626e+00,
          6.8525e-01,  4.2568e-01,  8.3430e-01, -1.0295e+00],
        [-1.6114e+00, -5.9811e-01, -6.2155e-01, -2.9091e-03,  5.5972e-01,
         -7.8317e-01, -8.3771e-02,  7.5206e-01, -1.4175e+00],
        [ 8.1634e-01,  4.9134e-01, -2.7166e-01, -6.0395e-01,  7.2573e-01,
          7.1407e-01,  1.7733e-01,  1.7713e-01,  1.4325e-01],
        [ 7.0480e-01, -1.6713e-01,  9.5935e-01, -7.8019e-01,  1.6251e+00,
          1.2835e+00,  4.8042e-02,  5.8192e-01, -3.0040e+00],
        [ 6.3264e-01,  2.4104e-01, -7.3922e-01,  4.7470e-01, -1.7250e+00,
         -7.5146e-01, -9.4407e-01, -8.6993e-01, -2.6355e+00],
        [-1.8073e-01, -1.2490e+00, -8.2327e-01,  1.4730e+00,  5.8836e-01,
          4.1034e-03,  9.9122e-01,  2.5957e-01, -1.8716e-01],
        [ 9.1802e-01, -3.6061e-01, -8.1788e-01,  1.5692e+00, -1.1461e+00,
         -5.7984e-01,  6.7999e-01, -4.1150e-02,  1.0829e+00],
        [ 1.1454e+00, -4.3676e-01,  2.2854e+00,  1.1893e+00,  4.0966e-01,
         -1.1711e-01, -2.1936e-02, -8.4547e-01,  1.2236e-01],
        [ 1.4835e+00, -1.6104e-01,  4.8580e-02, -2.6155e+00, -1.4138e-01,
          1.0443e+00, -2.3830e-01, -1.4911e+00,  3.5307e-01],
        [ 2.0946e+00,  1.9337e+00, -2.6824e-01,  2.3089e-01,  1.1514e-03,
          2.1862e+00,  6.3311e-01, -6.1647e-01, -1.3749e+00],
        [ 1.5384e+00, -7.6717e-01, -7.3752e-01,  1.2570e+00, -5.8681e-01,
          1.3887e+00, -9.6056e-01,  4.8157e-01, -4.1506e-01],
        [ 1.1072e-01,  1.1431e+00,  2.0399e+00, -5.5736e-01,  6.7684e-01,
```

```
 -6.8097e-01, -8.7569e-01, -1.2483e+00,  7.7616e-01],
[-2.6649e-01,  7.9188e-01,  7.2701e-01,  1.7280e+00, -1.1796e+00,
  5.2148e-01, -6.1184e-01,  3.1035e-01,  9.7009e-01],
[-8.3095e-01, -1.6064e+00,  2.3667e+00, -1.2204e+00,  4.1136e-01,
 -1.4684e+00,  6.3564e-02, -1.5051e+00, -2.2001e-01],
[-5.2230e-01,  8.1375e-01,  6.1553e-01, -4.2599e-02,  1.7301e-01,
 -1.2271e-01, -2.0114e+00, -7.8907e-01, -1.2734e+00],
[ 6.8031e-01, -3.0871e-01, -3.0772e-01,  6.3263e-01,  1.5590e+00,
  2.7520e-01, -1.0685e+00,  2.7201e-01,  9.9093e-01],
[ 9.2734e-02, -1.3574e+00,  4.0598e-01,  4.9756e-01,  5.6375e-01,
 -1.1143e+00,  5.2196e-01,  1.4329e-01, -3.9328e-01],
[ 2.9002e-01,  2.9804e-01,  2.2331e+00,  2.1968e+00, -5.0351e-01,
  1.5336e-01,  4.6150e-02,  1.7698e+00, -2.8478e-01],
[ 1.2679e+00, -5.9689e-01, -1.3967e+00,  1.0058e+00,  2.8197e-01,
  1.0780e+00, -5.4874e-01, -5.0334e-01, -2.8814e-01],
[ 2.1569e+00,  6.2626e-01,  1.9641e-01, -1.5439e+00,  5.7199e-01,
 -9.1998e-01, -1.1759e+00, -7.0328e-01,  3.1153e-01],
[-2.3782e-01,  1.5635e+00, -9.9518e-01,  3.7845e-01, -1.5209e+00,
 -7.1817e-01,  3.5974e-01,  1.7197e-01,  3.6362e-01],
[ 2.2835e-01, -1.5459e+00, -9.5902e-01, -1.1907e+00, -3.7331e-01,
  5.8753e-01, -1.4671e+00, -3.0594e-01,  8.4065e-01],
[ 1.1026e+00, -8.7721e-01,  1.5426e+00,  4.4972e-01,  3.1010e-01,
  9.7017e-01,  7.8906e-01,  1.0092e+00,  2.0941e+00],
[-1.7965e+00,  1.8701e-01,  4.3996e-01,  4.6749e-01,  4.0462e-01,
  9.2346e-02, -1.9932e+00,  7.7445e-01,  7.0330e-01],
[ 1.0255e+00,  1.0959e+00,  2.6846e-01, -1.3541e+00, -6.1543e-02,
 -3.2624e-01, -7.2100e-01, -4.0800e-01,  9.1167e-01],
[ 1.0763e+00, -1.4209e+00, -5.9040e-01, -1.2943e+00,  2.4534e-01,
 -9.7199e-01, -1.7028e+00, -1.6392e+00, -1.0653e+00],
[ 2.0118e-01, -2.0262e+00,  1.3364e+00, -1.7834e+00,  6.5608e-01,
 -1.3171e+00,  8.7381e-01, -2.6745e-01,  1.3398e-01],
[ 8.5316e-01,  8.4795e-01, -1.1855e-01, -3.7457e-02,  2.4010e-01,
  2.7003e-01,  1.1791e+00,  1.0496e+00,  1.6055e+00],
[-1.3566e+00, -8.5464e-01,  8.9450e-01,  1.3328e+00,  2.9470e-01,
 -6.0269e-01, -9.7720e-01,  4.0491e-01,  1.5337e+00],
[ 1.5095e+00,  2.3863e-01,  1.1418e+00, -7.8052e-01, -2.4659e-01,
 -8.4775e-01,  3.0722e-01,  6.8697e-01,  8.0159e-02],
[-1.3932e+00, -6.0093e-01, -8.3311e-01,  2.6036e-01, -6.4276e-01,
  3.4897e-01, -1.7955e+00, -7.9129e-01,  1.5356e-01]])
```

[10]:
```
# En multipliant ces "poids" par nos vecteurs one-hot organisés en matrice...
# On obtient des valeurs que l'on va "interpréter" comme des logs (log-counts).
# En utilisant l'exponentielle de ces valeurs, on va retrouver quelque chose
# d'équivalent à la matrice N que nous avions définie précédemment dans la␣
 ↪méthode
# par comptage.
xenc @ W
```

[10]:
```
tensor([[ 0.9184, -1.1502, -0.8363,  1.5226,  0.8434,  0.3710,  1.8180, -0.6987,
          1.1443],
        [ 0.2640, -0.1092,  0.5762,  0.1440,  0.7639,  0.3053, -0.7356, -0.9621,
          0.3534],
        [ 0.1261,  0.8596, -0.7190, -2.6217,  0.4382,  2.3034, -1.7862,  0.3502,
          0.9571],
        [ 0.1261,  0.8596, -0.7190, -2.6217,  0.4382,  2.3034, -1.7862,  0.3502,
          0.9571],
        [ 0.6582, -0.7624, -0.0851, -1.5097,  0.3336, -0.0473, -0.5248,  1.1375,
         -0.3833],
        [ 1.1454, -0.4368,  2.2854,  1.1893,  0.4097, -0.1171, -0.0219, -0.8455,
          0.1224],
        [ 0.1107,  1.1431,  2.0399, -0.5574,  0.6768, -0.6810, -0.8757, -1.2483,
          0.7762],
        [ 1.1026, -0.8772,  1.5426,  0.4497,  0.3101,  0.9702,  0.7891,  1.0092,
          2.0941],
        [ 0.6582, -0.7624, -0.0851, -1.5097,  0.3336, -0.0473, -0.5248,  1.1375,
         -0.3833]])
```

[11]:
```
logits = xenc @ W # log-counts
counts = logits.exp() # statut équivalent à N
probs = counts / counts.sum(1, keepdims=True) # distribution de probabilités␣
 ↪(equ. à p)
probs
```

[11]:
```
tensor([[0.1170, 0.0148, 0.0202, 0.2141, 0.1086, 0.0677, 0.2877, 0.0232,
0.1467],
        [0.1192, 0.0821, 0.1629, 0.1057, 0.1965, 0.1243, 0.0439, 0.0350,
0.1304],
        [0.0573, 0.1193, 0.0246, 0.0037, 0.0783, 0.5053, 0.0085, 0.0717,
0.1315],
        [0.0573, 0.1193, 0.0246, 0.0037, 0.0783, 0.5053, 0.0085, 0.0717,
0.1315],
        [0.1879, 0.0454, 0.0893, 0.0215, 0.1358, 0.0928, 0.0576, 0.3034,
0.0663],
        [0.1439, 0.0296, 0.4501, 0.1504, 0.0690, 0.0407, 0.0448, 0.0197,
0.0518],
        [0.0625, 0.1756, 0.4304, 0.0321, 0.1101, 0.0283, 0.0233, 0.0161,
0.1216],
        [0.1126, 0.0156, 0.1749, 0.0586, 0.0510, 0.0987, 0.0823, 0.1026,
0.3036],
        [0.1879, 0.0454, 0.0893, 0.0215, 0.1358, 0.0928, 0.0576, 0.3034,
0.0663]])
```

Réseau de neurones sur cet exemple

[12]:
```
# Initialisation de "nb_chars" poids de neurones
g = torch.Generator().manual_seed(2147483647)
```

```
W = torch.randn((nb_chars, nb_chars), generator=g, requires_grad=True)
```

[13]:
```
# Réseau à une couche (probs)
xenc = F.one_hot(xs, num_classes=nb_chars).float() # input to the network:␣
 ↪one-hot encoding
logits = xenc @ W  # predict log-counts
counts = logits.exp() # counts, equivalent to N
probs = counts / counts.sum(1, keepdims=True)  # probabilities for next␣
 ↪character
# btw: the last 2 lines here are together called a 'softmax'
```

[14]:
```
nlls = torch.zeros(5)
for i in range(5):
    # i-th bigram:
    x = xs[i].item() # input character index
    y = ys[i].item() # label character index
    print('--------')
    print(f'bigram example {i+1}: {itoc[x]}{itoc[y]} (indexes {x},{y})')
    print('input to the neural net:', x)
    print('output probabilities from the neural net:', probs[i])
    print('label (actual next character):', y)
    p = probs[i, y]
    print('probability assigned by the net to the the correct character:', p.
 ↪item())
    logp = torch.log(p)
    print('log likelihood:', logp.item())
    nll = -logp
    print('negative log likelihood:', nll.item())
    nlls[i] = nll

print('=========')
print('average negative log likelihood, i.e. loss =', nlls.mean().item())
```

```
--------
bigram example 1: .a (indexes 0,3)
input to the neural net: 0
output probabilities from the neural net: tensor([0.0495, 0.0081, 0.0100,
0.0034, 0.0137, 0.0100, 0.0022, 0.0189, 0.0112,
        0.0255, 0.0064, 0.0227, 0.0074, 0.0067, 0.0407, 0.1939, 0.0492, 0.0020,
        0.0203, 0.0045, 0.0276, 0.0089, 0.0023, 0.0162, 0.0096, 0.1253, 0.1189,
        0.0053, 0.0030, 0.0140, 0.0035, 0.0214, 0.0109, 0.0382, 0.0046, 0.0044,
        0.0017, 0.0361, 0.0030, 0.0348, 0.0039], grad_fn=<SelectBackward0>)
label (actual next character): 3
probability assigned by the net to the the correct character:
0.003431369084864855
log likelihood: -5.674796104431152
negative log likelihood: 5.674796104431152
--------
```

```
bigram example 2: ac (indexes 3,5)
input to the neural net: 3
output probabilities from the neural net: tensor([0.0017, 0.0064, 0.0258,
0.0032, 0.0085, 0.0247, 0.0371, 0.0103, 0.0104,
        0.0024, 0.0027, 0.0207, 0.0226, 0.0620, 0.0193, 0.0406, 0.1549, 0.0225,
        0.0073, 0.0261, 0.0076, 0.0234, 0.0546, 0.0178, 0.0089, 0.0141, 0.0084,
        0.0245, 0.0226, 0.0035, 0.0713, 0.0167, 0.0378, 0.0234, 0.0390, 0.0021,
        0.0092, 0.0017, 0.0368, 0.0490, 0.0181], grad_fn=<SelectBackward0>)
label (actual next character): 5
probability assigned by the net to the the correct character:
0.024747123941779137
log likelihood: -3.6990458965301514
negative log likelihood: 3.6990458965301514
--------
bigram example 3: cc (indexes 5,5)
input to the neural net: 5
output probabilities from the neural net: tensor([0.0355, 0.0383, 0.0081,
0.0299, 0.0039, 0.0132, 0.0410, 0.0156, 0.0065,
        0.0104, 0.0081, 0.0344, 0.0140, 0.0583, 0.0237, 0.0640, 0.0057, 0.0073,
        0.0205, 0.0083, 0.0115, 0.0048, 0.0567, 0.0031, 0.0105, 0.0059, 0.0147,
        0.0128, 0.0543, 0.0087, 0.0325, 0.0304, 0.1164, 0.0116, 0.0171, 0.0283,
        0.0362, 0.0034, 0.0144, 0.0673, 0.0128], grad_fn=<SelectBackward0>)
label (actual next character): 5
probability assigned by the net to the the correct character:
0.013233082368969917
log likelihood: -4.325035572052002
negative log likelihood: 4.325035572052002
--------
bigram example 4: ce (indexes 5,7)
input to the neural net: 5
output probabilities from the neural net: tensor([0.0355, 0.0383, 0.0081,
0.0299, 0.0039, 0.0132, 0.0410, 0.0156, 0.0065,
        0.0104, 0.0081, 0.0344, 0.0140, 0.0583, 0.0237, 0.0640, 0.0057, 0.0073,
        0.0205, 0.0083, 0.0115, 0.0048, 0.0567, 0.0031, 0.0105, 0.0059, 0.0147,
        0.0128, 0.0543, 0.0087, 0.0325, 0.0304, 0.1164, 0.0116, 0.0171, 0.0283,
        0.0362, 0.0034, 0.0144, 0.0673, 0.0128], grad_fn=<SelectBackward0>)
label (actual next character): 7
probability assigned by the net to the the correct character:
0.01563512347638607
log likelihood: -4.158235549926758
negative log likelihood: 4.158235549926758
--------
bigram example 5: ep (indexes 7,17)
input to the neural net: 7
output probabilities from the neural net: tensor([0.0579, 0.0032, 0.0506,
0.0075, 0.0371, 0.0182, 0.0344, 0.0079, 0.0179,
        0.0157, 0.0043, 0.0297, 0.0035, 0.0061, 0.0524, 0.0082, 0.0414, 0.0450,
        0.0362, 0.0097, 0.0044, 0.0414, 0.0829, 0.0135, 0.0096, 0.0334, 0.0228,
```

```
        0.0134, 0.0243, 0.0257, 0.0424, 0.0110, 0.0411, 0.0228, 0.0270, 0.0017,
        0.0121, 0.0113, 0.0182, 0.0468, 0.0071], grad_fn=<SelectBackward0>)
label (actual next character): 17
probability assigned by the net to the the correct character:
0.04502563923597336
log likelihood: -3.1005232334136963
negative log likelihood: 3.1005232334136963
=========
average negative log likelihood, i.e. loss = 4.191527366638184
```

Optimization sur un mot

[36]:
```
# forward pass
xenc = F.one_hot(xs, num_classes=nb_chars).float() # input to the network:␣
 ↪one-hot encoding
logits = xenc @ W # predict log-counts
counts = logits.exp() # counts, equivalent to N
probs = counts / counts.sum(1, keepdims=True) # probabilities for next character
loss = -probs[torch.arange(tensor_dims), ys].log().mean()
```

[37]:
```
print(loss.item())
```

```
3.725316047668457
```

[38]:
```
# backward pass
W.grad = None # set to zero the gradient
loss.backward()
```

[39]:
```
W.data += -0.1 * W.grad
# ^^ loop above from forward pass and see loss decreasing
```

1.2.2 Synthèse: apprentissage complet

[40]:
```
#
# Générateur des mots selon notre modèle de langue génératif bigrams par réseau␣
 ↪de neurones
#
import torch

# Lecture des données
EOS='.'
words = open('civil_mots.txt', 'r').read().splitlines()
chars = sorted(list(set(''.join(words))))
nb_chars = len(chars) + 1  # On ajoute 1 pour EOS

# Dictionnaires caractère <-> entier
ctoi = {c:i+1 for i,c in enumerate(chars)}
ctoi['.'] = 0
itoc = {i:c for c,i in ctoi.items()}# Création du dataset avec tous les mots
```

```
# Génération du jeu d'entraînement
xs, ys = [], []
for w in words:
    chs = ['.'] + list(w) + ['.']
    for ch1, ch2 in zip(chs, chs[1:]):
        ix1 = ctoi[ch1]
        ix2 = ctoi[ch2]
        xs.append(ix1)
        ys.append(ix2)
xs = torch.tensor(xs)
ys = torch.tensor(ys)
num = xs.nelement()
print('NB exemples:', num)

# Initialisation du réseau (une seule couche de neurones sans biais)
g = torch.Generator().manual_seed(2147483647)
W = torch.randn((nb_chars, nb_chars), generator=g, requires_grad=True)
```

```
NB exemples: 67652
```

[41]:
```
# Apprentissage: descente du gradient
for k in range(600):

  # Forward pass
  xenc = F.one_hot(xs, num_classes=nb_chars).float() # input to the network: one-hot encoding
  logits = xenc @ W # predict log-counts (logits)
  counts = logits.exp() # counts, equivalent to N
  probs = counts / counts.sum(1, keepdims=True) # probabilities for next character
  loss = -probs[torch.arange(num), ys].log().mean() + 0.01*(W**2).mean()  # + 0.01... for smoothing the model
  print(loss.item())

  # backward pass
  W.grad = None # set to zero the gradient
  loss.backward()

  # update
  W.data += -50 * W.grad
```

```
4.268752574920654
3.8354008197784424
3.5137295722961426
3.2882258892059326
3.136672258377075
3.0282742977142334
```

2.9461023807525635
2.881960153579712
2.830700159072876
2.788752555847168
2.75366473197937
2.723792552947998
2.69801926612854
2.6755597591400146
2.6558377742767334
2.6384148597717285
2.622948169708252
2.609158754348755
2.5968127250671387
2.585711717605591
2.575685501098633
2.566587209701538
2.5582938194274902
2.5506985187530518
2.543713331222534
2.537261962890625
2.5312814712524414
2.5257174968719482
2.5205249786376953
2.5156655311584473
2.5111048221588135
2.5068154335021973
2.5027718544006348
2.498953342437744
2.495340585708618
2.491917610168457
2.4886693954467773
2.4855830669403076
2.482647657394409
2.4798519611358643
2.477186679840088
2.4746437072753906
2.472215175628662
2.4698939323425293
2.4676730632781982
2.4655468463897705
2.463510274887085
2.461557626724243
2.459684371948242
2.457886219024658
2.4561588764190674
2.454498529434204
2.452902317047119
2.4513659477233887

2.4498867988586426
2.4484617710113525
2.4470887184143066
2.4457643032073975
2.4444868564605713
2.443253517150879
2.4420626163482666
2.4409120082855225
2.4397995471954346
2.4387240409851074
2.43768310546875
2.436675786972046
2.4357004165649414
2.434755325317383
2.4338393211364746
2.4329514503479004
2.4320902824401855
2.4312546253204346
2.430443286895752
2.4296555519104004
2.4288904666900635
2.4281468391418457
2.4274234771728516
2.4267208576202393
2.4260365962982178
2.425370693206787
2.424722671508789
2.424091339111328
2.423476219177246
2.422877311706543
2.422293186187744
2.4217233657836914
2.4211676120758057
2.4206252098083496
2.4200961589813232
2.419579029083252
2.419074296951294
2.418581247329712
2.4180994033813477
2.417628526687622
2.4171676635742188
2.416717529296875
2.4162769317626953
2.4158456325531006
2.415423631668091
2.415010690689087
2.4146060943603516
2.4142098426818848

2.4138216972351074
2.4134411811828613
2.4130687713623047
2.412703275680542
2.4123449325561523
2.4119937419891357
2.411648750305176
2.4113104343414307
2.4109785556793213
2.4106526374816895
2.4103331565856934
2.4100189208984375
2.4097108840942383
2.4094078540802
2.4091103076934814
2.408817768096924
2.4085307121276855
2.4082484245300293
2.407970905303955
2.407698392868042
2.4074296951293945
2.407166004180908
2.4069066047668457
2.406651258468628
2.406399965286255
2.4061529636383057
2.405909776687622
2.405670404434204
2.4054346084594727
2.405202627182007
2.4049742221832275
2.404749631881714
2.4045279026031494
2.4043097496032715
2.404094696044922
2.4038827419281006
2.403674364089966
2.4034688472747803
2.403266191482544
2.403066396713257
2.402869462966919
2.4026753902435303
2.402484178543091
2.4022953510284424
2.4021096229553223
2.401926040649414
2.401745080947876
2.401566743850708

2.401390790939331
2.401216983795166
2.40104603767395
2.400876998901367
2.400709867477417
2.400545597076416
2.4003829956054688
2.4002227783203125
2.40006422996521
2.3999080657958984
2.399754047393799
2.399601936340332
2.39945125579834
2.3993031978607178
2.3991565704345703
2.3990116119384766
2.3988687992095947
2.3987278938293457
2.3985884189605713
2.3984506130218506
2.3983144760131836
2.3981800079345703
2.3980472087860107
2.397916078567505
2.3977863788604736
2.397658348083496
2.3975319862365723
2.3974063396453857
2.3972833156585693
2.3971610069274902
2.397040367126465
2.396921157836914
2.396803140640259
2.396686315536499
2.396571159362793
2.3964574337005615
2.3963444232940674
2.396233320236206
2.396122932434082
2.3960142135620117
2.3959062099456787
2.3957996368408203
2.3956944942474365
2.395590305328369
2.3954873085021973
2.3953850269317627
2.395284414291382
2.3951847553253174

2.3950858116149902
2.394988536834717
2.3948915004730225
2.3947958946228027
2.3947017192840576
2.3946080207824707
2.3945152759552
2.394423723220825
2.3943333625793457
2.3942437171936035
2.3941547870635986
2.39406681060791
2.393979787826538
2.3938939571380615
2.393808364868164
2.393724203109741
2.3936407566070557
2.3935577869415283
2.3934762477874756
2.393394708633423
2.3933145999908447
2.393235206604004
2.3931565284729004
2.393078327178955
2.3930013179779053
2.3929250240325928
2.3928489685058594
2.3927741050720215
2.392699718475342
2.3926267623901367
2.3925535678863525
2.3924813270568848
2.3924098014831543
2.3923392295837402
2.392268657684326
2.3921995162963867
2.3921306133270264
2.3920624256134033
2.3919947147369385
2.391927719116211
2.3918612003326416
2.3917956352233887
2.3917300701141357
2.391665458679199
2.391601324081421
2.391538381576538
2.391475200653076
2.3914127349853516

2.3913512229919434
2.3912899494171143
2.3912291526794434
2.3911690711975098
2.391108989715576
2.391049861907959
2.390991449356079
2.3909332752227783
2.3908755779266357
2.3908183574676514
2.390761613845825
2.3907055854797363
2.3906497955322266
2.390594959259033
2.3905396461486816
2.3904852867126465
2.3904316425323486
2.390378475189209
2.3903253078460693
2.390272378921509
2.3902206420898438
2.3901686668395996
2.3901174068450928
2.390066623687744
2.3900163173675537
2.3899660110473633
2.38991641998291
2.389867067337036
2.3898181915283203
2.3897697925567627
2.389721632003784
2.389674186706543
2.3896265029907227
2.3895797729492188
2.389533042907715
2.38948655128479
2.3894405364990234
2.3893954753875732
2.389349937438965
2.389305353164673
2.38926100730896
2.389216899871826
2.3891727924346924
2.389129161834717
2.3890860080718994
2.3890435695648193
2.389000654220581
2.38895845413208

2.3889169692993164
2.3888752460479736
2.388834238052368
2.3887929916381836
2.3887522220611572
2.388712167739868
2.388671875
2.388632297515869
2.3885927200317383
2.3885533809661865
2.388514757156372
2.3884763717651367
2.3884377479553223
2.388399600982666
2.388362169265747
2.388324499130249
2.3882875442504883
2.3882505893707275
2.3882133960723877
2.3881771564483643
2.388140916824341
2.3881051540374756
2.3880693912506104
2.3880341053009033
2.3879990577697754
2.3879637718200684
2.3879292011260986
2.387895107269287
2.3878607749938965
2.387826681137085
2.3877930641174316
2.3877594470977783
2.387726306915283
2.387693405151367
2.387660503387451
2.387627601623535
2.3875954151153564
2.3875629901885986
2.387531280517578
2.3874995708465576
2.3874683380126953
2.387436628341675
2.3874058723449707
2.3873748779296875
2.3873443603515625
2.3873136043548584
2.3872838020324707
2.3872532844543457

2.387223720550537
2.3871941566467285
2.387164354324341
2.3871355056762695
2.38710618019104
2.3870770931243896
2.3870487213134766
2.3870201110839844
2.386991500854492
2.3869638442993164
2.3869357109069824
2.3869078159332275
2.3868801593780518
2.386852741241455
2.3868257999420166
2.386798858642578
2.3867716789245605
2.386744976043701
2.386718511581421
2.3866920471191406
2.3866658210754395
2.3866398334503174
2.386613368988037
2.3865880966186523
2.3865625858306885
2.3865370750427246
2.38651180267334
2.386486530303955
2.3864617347717285
2.386437177658081
2.3864123821258545
2.386388063430786
2.386363983154297
2.3863391876220703
2.3863158226013184
2.386291265487671
2.38626766204834
2.386244058609009
2.386220932006836
2.386197566986084
2.386174440383911
2.3861513137817383
2.3861281871795654
2.386105537414551
2.3860831260681152
2.3860607147216797
2.386038303375244
2.3860161304473877

2.3859941959381104
2.385972023010254
2.3859503269195557
2.3859286308288574
2.385906934738159
2.385885715484619
2.3858642578125
2.38584303855896
2.385822057723999
2.385801076889038
2.3857805728912354
2.3857595920562744
2.385739326477051
2.385718584060669
2.3856985569000244
2.385678291320801
2.385658025741577
2.3856382369995117
2.385618209838867
2.3855984210968018
2.3855788707733154
2.385559320449829
2.385540246963501
2.3855206966400146
2.3855013847351074
2.3854825496673584
2.3854634761810303
2.385444402694702
2.3854258060455322
2.385406970977783
2.3853886127471924
2.3853702545166016
2.3853516578674316
2.385333776473999
2.385315418243408
2.3852975368499756
2.385279655456543
2.3852617740631104
2.3852438926696777
2.385226249694824
2.38520884513855
2.385190963745117
2.385173797607422
2.3851566314697266
2.3851394653320312
2.385122537612915
2.3851053714752197
2.3850882053375244

2.3850717544555664
2.3850550651550293
2.385038137435913
2.385021686553955
2.3850057125091553
2.3849892616271973
2.38497257232666
2.3849565982818604
2.3849403858184814
2.3849244117736816
2.384908437728882
2.384892463684082
2.3848769664764404
2.3848612308502197
2.384845733642578
2.3848302364349365
2.384814739227295
2.3847994804382324
2.3847837448120117
2.3847687244415283
2.384753704071045
2.3847386837005615
2.3847239017486572
2.3847086429595947
2.3846943378448486
2.3846793174743652
2.38466477394104
2.384650468826294
2.3846359252929688
2.3846211433410645
2.3846068382263184
2.3845925331115723
2.3845784664154053
2.3845643997192383
2.384549856185913
2.3845362663269043
2.3845224380493164
2.3845083713531494
2.3844945430755615
2.3844809532165527
2.384467124938965
2.384453773498535
2.3844399452209473
2.3844265937805176
2.384413480758667
2.384399890899658
2.3843865394592285
2.384373426437378

2.3843603134155273
2.384347438812256
2.3843343257904053
2.3843212127685547
2.384308338165283
2.3842954635620117
2.3842833042144775
2.384270429611206
2.3842577934265137
2.3842451572418213
2.384232759475708
2.3842198848724365
2.3842077255249023
2.384195327758789
2.384183168411255
2.3841710090637207
2.3841586112976074
2.3841466903686523
2.384134531021118
2.384122848510742
2.384110450744629
2.384098768234253
2.384087085723877
2.384075164794922
2.384063482284546
2.38405179977417
2.384040355682373
2.384028673171997
2.3840174674987793
2.3840057849884033
2.3839943408966064
2.3839831352233887
2.383971929550171
2.383960485458374
2.383949041366577
2.3839383125305176
2.383927345275879
2.3839163780212402
2.3839049339294434
2.3838939666748047
2.383883476257324
2.3838727474212646
2.383861780166626
2.3838508129119873
2.383840322494507
2.3838295936584473
2.383819341659546
2.3838088512420654

2.383798122406006
2.3837876319885254
2.383777141571045
2.3837666511535645
2.383756399154663
2.383746385574341
2.3837358951568604
2.383725643157959
2.3837156295776367
2.3837058544158936
2.383695602416992
2.383685350418091
2.3836755752563477
2.3836655616760254
2.383655548095703
2.38364577293396
2.383636236190796
2.383626699447632
2.3836166858673096
2.3836071491241455
2.3835973739624023
2.3835878372192383
2.383578300476074
2.38356876373291
2.383559465408325
2.383549928665161
2.383540630340576
2.383531332015991
2.3835220336914062
2.3835127353668213
2.3835034370422363
2.3834943771362305
2.3834853172302246
2.3834760189056396
2.383467197418213
2.383457899093628
2.383448600769043
2.3834400177001953
2.3834309577941895
2.383422374725342
2.383413553237915
2.383404493331909
2.3833956718444824
2.3833870887756348
2.383378505706787
2.3833696842193604
2.3833611011505127
2.383352279663086

```
2.3833439350128174
2.3833351135253906
2.383326768875122
2.3833186626434326
2.3833096027374268
2.3833014965057373
2.3832931518554688
2.3832848072052
2.3832764625549316
2.383268117904663
2.3832600116729736
2.383251667022705
2.3832435607910156
2.383235454559326
2.3832273483276367
2.3832192420959473
2.383211135864258
2.3832032680511475
```

[42]:
```python
# finally, sample from the 'neural net' model
g = torch.Generator().manual_seed(2147483647)

for i in range(5):

  out = []
  ix = 0
  while True:

    # ----------
    # Avec l'approche par comptage on utilisait:
    #p = P[ix]
    # ----------
    # NOW:
    xenc = F.one_hot(torch.tensor([ix]), num_classes=nb_chars).float()
    logits = xenc @ W # predict log-counts
    counts = logits.exp() # counts, equivalent to N
    p = counts / counts.sum(1, keepdims=True) # probabilities for next character
    # ----------

    ix = torch.multinomial(p, num_samples=1, replacement=True, generator=g).
 ↪item()
    out.append(itoc[ix])
    if ix == 0:
      break
  print(''.join(out))
```

```
éssanée.
mexcororér.
```

```
monts.
ex.
moût.
```

On voit qu'on a les mêmes mots que ceux générés par comptage, nous avons donc bâti une méthode neuronale équivalente à ce qu'on obtient par la méthode par comptage.