InfoNCE Loss Psuedo

November 23, 2021

[30]:
```
import torch
import numpy as  np
import random
from torch import Tensor
from torch import nn
from typing import List
from torch import exp
from torch import log
from torch.nn import LogSoftmax
from torch import diag
from torch import arange
```

[2]:
```
t = torch.empty((30,20))
t.shape
```

[2]:
```
torch.Size([30, 20])
```

[7]:
```
def info_NCE(X:Tensor, W:List[Tensor], encoder:nn.Module, autoregressive:nn.
↪Module):
    batch, channels, data_length = X.shape #eg. (64 x 12 x 4500)

    latent_matrix = encoder(X)
    batch, timesteps, latent_size = latent_matrix.shape #eg. (64 x 27 x 128)

    context_matrix = autoregressive(latent_matrix)
    batch, timesteps, context_size = context_matrix.shape #eg. (64 x 27 x 256)

    k = len(W) #eg. 12
    current_timestep = timesteps-k-1
    last_context_vector = context_matrix[:, current_timestep, :] #shape: (64 x␣
↪256)
    loss = 0.
    for i in range(batch):
        loss_i = 0
        for step in range(k):
            latent_vector_pred_i = last_context_vector[i] @ W[step] #shape:␣
↪(256) @ (256 x 128) = (128)
```

```
            latent_vector_i = latent_matrix[i, current_timestep+step+1, :]␣
↪#shape: (128)
            p_xi_ct = exp(latent_vector_pred_i @ latent_vector_i) #scalar
            denominator = 0.
            for j in range(batch):
                latent_vector_j = latent_matrix[j, current_timestep+step+1, :]␣
↪#shape: (128)
                p_xj_ct = exp(latent_vector_pred_i @ latent_vector_j) #scalar
                denominator += p_xj_ct
            loss_i += log(p_xi_ct/denominator)
        loss += loss_i/-k
    return loss / batch
```

[105]:
```
def info_NCE(X:Tensor, W:List[Tensor], encoder:nn.Module, autoregressive:nn.
↪Module):
    batch, channels, data_length = X.shape #eg. (64 x 12 x 4500)

    latent_matrix = encoder(X)
    batch, timesteps, latent_size = latent_matrix.shape #eg. (64 x 27 x 128)

    context_matrix = autoregressive(latent_matrix)
    batch, timesteps, context_size = context_matrix.shape #eg. (64 x 27 x 256)

    k = len(W) #eg. 12
    current_timestep = timesteps-k-1
    last_context_vector = context_matrix[:, current_timestep, :] #shape: (64 x␣
↪256)
    loss = 0.
    accuracy = 0.
    for t in range(k):
        #W[t].shape eg. 256 x 128
        latent_vector = latent_matrix[:, current_timestep+t+1, :] #shape: (64 x␣
↪128)
        latent_vector_pred = last_context_vector @ W[t] #shape: (64 x 128)

        scalar_sim = latent_vector_pred @ latent_vector.T #shape (64 x 64)
        softmax = LogSoftmax(dim=1)(scalar_sim) #shape (64 x 64)
        p_xi_c_t = diag(softmax) #shape (64)
        loss += p_xi_c_t.sum() #scalar
        accuracy += (softmax.argmax(dim=1) == arange(batch)).sum() #scalar
    loss /= -k*batch
    accuracy /= k*batch
    return loss, accuracy
```

[97]:
```
encoder = lambda x: torch.randn((64, 27, 128))
auto = lambda x: torch.randn((64, 27, 256))
W = [torch.ones((256, 128))]*12
```

[104]:
```
X = torch.empty((64, 12, 4500))
loss = info_NCE(X, W, encoder, auto)
loss
```

[104]:
```
(tensor(-0.), tensor(1.))
```

[75]:
```
latent()
```

[75]:
```
tensor([0, 2, 0, 1, 1, 1, 1, 0, 0, 0, 0, 2])
```

[28]:
```
soft = torch.nn.Softmax(dim=0)
s = soft(encoder(1)[:, 0]@encoder(1)[:, 0].T)
s[:, 0].sum()
```

[28]:
```
tensor(1.)
```

[107]:
```
l = np.arange(12).repeat(4).reshape(12,4) #shape 12x4
lp = np.ones((12,4))
```

[109]:
```
l @ lp.T
```

[109]:
```
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.,  4.],
       [ 8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.,  8.],
       [12., 12., 12., 12., 12., 12., 12., 12., 12., 12., 12., 12.],
       [16., 16., 16., 16., 16., 16., 16., 16., 16., 16., 16., 16.],
       [20., 20., 20., 20., 20., 20., 20., 20., 20., 20., 20., 20.],
       [24., 24., 24., 24., 24., 24., 24., 24., 24., 24., 24., 24.],
       [28., 28., 28., 28., 28., 28., 28., 28., 28., 28., 28., 28.],
       [32., 32., 32., 32., 32., 32., 32., 32., 32., 32., 32., 32.],
       [36., 36., 36., 36., 36., 36., 36., 36., 36., 36., 36., 36.],
       [40., 40., 40., 40., 40., 40., 40., 40., 40., 40., 40., 40.],
       [44., 44., 44., 44., 44., 44., 44., 44., 44., 44., 44., 44.]])
```

[110]:
```
lp @ l.T
```

[110]:
```
array([[ 0.,  4.,  8., 12., 16., 20., 24., 28., 32., 36., 40., 44.],
       [ 0.,  4.,  8., 12., 16., 20., 24., 28., 32., 36., 40., 44.],
       [ 0.,  4.,  8., 12., 16., 20., 24., 28., 32., 36., 40., 44.],
       [ 0.,  4.,  8., 12., 16., 20., 24., 28., 32., 36., 40., 44.],
       [ 0.,  4.,  8., 12., 16., 20., 24., 28., 32., 36., 40., 44.],
       [ 0.,  4.,  8., 12., 16., 20., 24., 28., 32., 36., 40., 44.],
       [ 0.,  4.,  8., 12., 16., 20., 24., 28., 32., 36., 40., 44.],
```

```
         [ 0.,  4.,  8., 12., 16., 20., 24., 28., 32., 36., 40., 44.],
         [ 0.,  4.,  8., 12., 16., 20., 24., 28., 32., 36., 40., 44.],
         [ 0.,  4.,  8., 12., 16., 20., 24., 28., 32., 36., 40., 44.],
         [ 0.,  4.,  8., 12., 16., 20., 24., 28., 32., 36., 40., 44.],
         [ 0.,  4.,  8., 12., 16., 20., 24., 28., 32., 36., 40., 44.]])
```

[69]:
```
s = torch.nn.Softmax(dim=1)(torch.Tensor(lp @ l.T))
s
```

[69]:
```
tensor([[7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
          2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02,
9.8168e-01],
        [7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
          2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02,
9.8168e-01],
        [7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
          2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02,
9.8168e-01],
        [7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
          2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02,
9.8168e-01],
        [7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
          2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02,
9.8168e-01],
        [7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
          2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02,
9.8168e-01],
        [7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
          2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02,
9.8168e-01],
        [7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
          2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02,
9.8168e-01],
        [7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
          2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02,
9.8168e-01],
        [7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
          2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02,
9.8168e-01],
        [7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
          2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02,
9.8168e-01],
        [7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
          2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02,
9.8168e-01]])
```

[70]:
```
s[:, 0] #same latent other, prediction
```

[70]:
```
tensor([7.6386e-20, 7.6386e-20, 7.6386e-20, 7.6386e-20, 7.6386e-20, 7.6386e-20,
        7.6386e-20, 7.6386e-20, 7.6386e-20, 7.6386e-20, 7.6386e-20, 7.6386e-20])
```

[71]:
```
s[0, :] #same prediction, other latent
```

[71]:
```
tensor([7.6386e-20, 4.1705e-18, 2.2770e-16, 1.2432e-14, 6.7878e-13, 3.7060e-11,
        2.0234e-09, 1.1047e-07, 6.0317e-06, 3.2932e-04, 1.7980e-02, 9.8168e-01])
```

[ ]:
```

```