Aula_7_Complete_Self_Attention_with_Good_Practices_(Rafael_Ito)

April 23, 2020

1 Notebook de referência

Usar as secções como guia.

Nome: Rafael Ito

Neste colab iremos treinar um modelo para fazer análise de sentimento usando o dataset IMDB.

Installing packages

[0]:
```
!pip install -q \
    numpy       \
    torch       \
    sklearn     \
    skorch      \
    matplotlib  \
    tqdm        \
    pytorch_lightning
```

Habilitamos o linting (avisa sobre erros de formatação no código)

[0]:
```
#!pip install --quiet flake8-nb pycodestyle_magic
#%load_ext pycodestyle_magic
#%flake8_on
##%flake8_off
```

Importing libraries

[3]:
```
#-------------------------------------------------
# general
#-------------------
import numpy as np
#import pandas as pd
import pytorch_lightning as pl
from multiprocessing import cpu_count
#----------------------------
import pdb
# pdb.set_trace() # breakpoint
#-------------------------------------------------
# PyTorch
#-------------------
import torch
```

```python
from torch import optim
from torch.nn import CrossEntropyLoss
from torch.utils.data import TensorDataset
from torch.utils.data import Dataset
from torchtext.vocab import GloVe
import torch.nn.functional as F
from torch.nn import Module
from torch.nn import Linear
from torch.nn import Dropout
from torch.nn import LayerNorm
from torch.nn import Embedding
from torch.nn import Sequential
#-------------------------------------------------
# skorch
#-------------------
#from skorch import NeuralNetClassifier
#-------------------------------------------------
# scikit-learn
#-------------------
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
#-------------------------------------------------
# data visualization
#-------------------
import matplotlib.pyplot as plt
import seaborn as sns
#-------------------------------------------------
# additional config
#-------------------
# random seed generator
np.random.seed(42)
torch.manual_seed(42);
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

Descobrimos se há uma GPU disponível

[4]:
```python
if torch.cuda.is_available():
   dev = "cuda:0"
else:
   dev = "cpu"
print(dev)
device = torch.device(dev)
#----------------------------
# get GPU model used
GPU_model = torch.cuda.get_device_name(0)
print('GPU model:', GPU_model)
```

```
cuda:0
```

```
GPU model: Tesla P100-PCIE-16GB
```

1.1 Preparando Dados

Primeiro, fazemos download do dataset:

[5]:
```
!wget -nc http://files.fast.ai/data/aclImdb.tgz
!tar -xzf aclImdb.tgz
```

```
File ‘aclImdb.tgz’ already there; not retrieving.
```

1.2 Carregando o dataset

Criaremos uma divisão de treino (80%) e dev (20%) artificialmente.

Nota: Evitar de olhar ao máximo o dataset de teste para não ficar enviseado no que será testado. Em aplicações reais, o dataset de teste só estará disponível no futuro, ou seja, é quando o usuário começa a testar o seu produto.

[6]:
```
import os
import random


def load_texts(folder):
    texts = []
    for path in os.listdir(folder):
        with open(os.path.join(folder, path)) as f:
            texts.append(f.read())
    return texts

x_train_pos = load_texts('aclImdb/train/pos')
x_train_neg = load_texts('aclImdb/train/neg')
x_test_pos = load_texts('aclImdb/test/pos')
x_test_neg = load_texts('aclImdb/test/neg')

x_train = x_train_pos + x_train_neg
x_test = x_test_pos + x_test_neg
y_train = [True] * len(x_train_pos) + [False] * len(x_train_neg)
y_test = [True] * len(x_test_pos) + [False] * len(x_test_neg)

# Embaralhamos o treino para depois fazermos a divisão treino/dev.
c = list(zip(x_train, y_train))
random.shuffle(c)
x_train, y_train = zip(*c)

n_train = int(0.8 * len(x_train))

x_dev = x_train[n_train:]
y_dev = y_train[n_train:]
x_train = x_train[:n_train]
y_train = y_train[:n_train]

print(len(x_train), 'amostras de treino.')
```

```
print(len(x_dev), 'amostras de desenvolvimento.')
print(len(x_test), 'amostras de teste.')

print('3 primeiras amostras treino:')
for x, y in zip(x_train[:3], y_train[:3]):
    print(y, x[:100])

print('3 últimas amostras treino:')
for x, y in zip(x_train[-3:], y_train[-3:]):
    print(y, x[:100])

print('3 primeiras amostras dev:')
for x, y in zip(x_dev[:3], y_test[:3]):
    print(y, x[:100])

print('3 últimas amostras dev:')
for x, y in zip(x_dev[-3:], y_dev[-3:]):
    print(y, x[:100])
```

```
20000 amostras de treino.
5000 amostras de desenvolvimento.
25000 amostras de teste.
3 primeiras amostras treino:
True Hey now, yours truly, TheatreX, found this while grubbing through videos at
the flea market, in almo
False "That 'Malcom' show on FOX is really making a killing... can't we do our
own version?" I speculate a
True This show was appreciated by critics and those who realized that any
similarities between "Pushing D
3 últimas amostras treino:
True Rawhide was a wonderful TV western series. Focusing on a band of trail
drovers lead by the trail bos
True I loved this film when I was little. Today at 17 it is one of my all time
favorite animated films. B
True I've read some terrible things about this film, so I was prepared for the
worst. "Confusing. Muddled
3 primeiras amostras dev:
True This is one of those movies that you and a bunch of friends sit around
drinking beers, eating pizza,
True Oh man. If you want to give your internal Crow T. Robot a real workout,
this is the movie to pop int
True This film is awful. Not offensive but extremely predictable. The movie
follows the life of a small t
3 últimas amostras dev:
False (Rating: 21 by The Film Snob.) (See our blog What-To-See-Next for details
on our rating system.)<br
False I'm afraid that I didn't like this movie very much. Apart from a few
saving graces, it's nothing to
False IT was no sense and it was so awful... i think Hollywood have a lot of
film like that... you don't h
```

1.3 Download do word embedding

Lista dos modelos disponíveis: https://github.com/RaRe-Technologies/gensim-data#models

[7]:
```
import gensim.downloader as api

word2vec_model = api.load("glove-wiki-gigaword-300")
print('word2vec shape:', word2vec_model.vectors.shape)
```

```
INFO:summarizer.preprocessing.cleaner:'pattern' package not found; tag filters
are not available for English
INFO:gensim.models.utils_any2vec:loading projection weights from /root/gensim-
data/glove-wiki-gigaword-300/glove-wiki-gigaword-300.gz
/usr/local/lib/python3.6/dist-packages/smart_open/smart_open_lib.py:253:
UserWarning: This function is deprecated, use smart_open.open instead. See the
migration notes for details: https://github.com/RaRe-
Technologies/smart_open/blob/master/README.rst#migrating-to-the-new-open-
function
  'See the migration notes for details: %s' % _MIGRATION_NOTES_URL
INFO:gensim.models.utils_any2vec:loaded (400000, 300) matrix from /root/gensim-
data/glove-wiki-gigaword-300/glove-wiki-gigaword-300.gz
```

```
word2vec shape: (400000, 300)
```

1.4 Criando Vocabulário a partir do word embedding

[8]:
```
import itertools

vocab = {word: index for index, word in enumerate(word2vec_model.index2word)}

# Adicionando PAD token
vocab['[PAD]'] = len(vocab)
pad_vector = np.zeros((1, word2vec_model.vectors.shape[1]))
embeddings = np.concatenate((word2vec_model.vectors, pad_vector), axis=0)
# convert embeddings from numpy to pytorch float32
embeddings = torch.from_numpy(embeddings)
embeddings = torch.tensor(embeddings, dtype=torch.float32)

print('Número de palavras no vocabulário:', len(vocab))
print(f'20 tokens mais frequentes: {list(itertools.islice(vocab.keys(), 20))}')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:11: UserWarning: To
copy construct from a tensor, it is recommended to use
sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  # This is added back by InteractiveShellApp.init_path()
```

```
Número de palavras no vocabulário: 400001
20 tokens mais frequentes: ['the', ',', '.', 'of', 'to', 'and', 'in', 'a', '"',
"'s", 'for', '-', 'that', 'on', 'is', 'was', 'said', 'with', 'he', 'as']
```

1.5 Tokenizando o dataset e convertendo para índices (preferencialmente, usar o DataLoader)

[0]:
```
class MyDataset(Dataset):
    def __init__(self, texts, labels, vocab, seq_length=64):
        self.texts = texts
        self.labels = torch.tensor(labels, dtype=torch.int64)
        self.vocab = vocab
        self.seq_length = seq_length

        words_idx = self.tokens_to_ids_batch(self.texts, self.vocab)
        X_str, mask = self.truncate_and_pad(
                            batch_word_ids=words_idx,
                            pad_token_id=self.vocab['[PAD]'],
                            seq_length=self.seq_length)
        self.X = torch.tensor(X_str, dtype=torch.int64)
        self.mask = torch.tensor(mask, dtype=torch.int64)

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, index):
        return (self.X[index], self.labels[index], self.mask[index])

    def tokenize(self, texts):
        for char in ['"', '\'', '.', ',', ':', '-', '?', '!']:
            texts = texts.replace(char, ' ')
        return texts.lower().split()

    def tokens_to_ids(self, tokens, vocab):
        return [vocab[token] for token in tokens if token in vocab]

    def tokens_to_ids_batch(self, textss, vocab):
        return [self.tokens_to_ids(self.tokenize(texts), vocab) for texts in textss]

    def truncate_and_pad(self, batch_word_ids, pad_token_id, seq_length):
        batch_word_ids = [word_ids[:seq_length] for word_ids in batch_word_ids]
        mask = [
            [1] * len(word_ids) + [0] * (seq_length - len(word_ids))
            for word_ids in  batch_word_ids]
        batch_word_ids = [
            word_ids + [pad_token_id] * (seq_length - len(word_ids))
            for word_ids in batch_word_ids]
        return batch_word_ids, mask
```

[0]:
```
BATCH_SIZE = 128
```

1.6 Inicializando e testando o DataLoader

[11]:
```
from torch.utils.data import DataLoader

texts = ['we like pizza', 'he does not like apples']
labels = [0, 1]
```

```
mydataset_debug = MyDataset(
    texts=texts,
    labels=labels,
    vocab=vocab,
    #pad_token_id=vocab['[PAD]'],
    seq_length=100)

L_DEBUG = 100
B_DEBUG = 10
dataloader_debug = DataLoader(mydataset_debug, batch_size=10, shuffle=True,
                              num_workers=0)

batch_token_ids, batch_labels, batch_mask = next(iter(dataloader_debug))
print('batch_token_ids', batch_token_ids)
print('batch_labels', batch_labels)
print('batch_token_ids.shape:', batch_token_ids.shape)
print('batch_labels.shape:', batch_labels.shape)
```

```
batch_token_ids tensor([[    53,    117,   9388, 400000, 400000, 400000, 400000,
400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000],
        [    18,    260,     36,    117,  13134, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000, 400000,
         400000]])
batch_labels tensor([0, 1])
batch_token_ids.shape: torch.Size([2, 100])
batch_labels.shape: torch.Size([2])
```

dimensions

[13]:
```
V = len(vocab)
D = word2vec_model.vector_size
L = 200
H = 6
B = BATCH_SIZE
```

```
#---------------------------
print('V =', V)      # vocab size
print('D =', D)      # embedding dim
print('H =', H)      # length of sequence
print('L =', L)      # number of heads
print('B =', B)      # batch size
```

```
V = 400001
D = 300
H = 6
L = 200
B = 128
```

1.7 Definindo a Rede Neural

Multi-Head Attention Layer

```
[0]: class MultiHead(torch.nn.Module):
         def __init__(self, L, D, H):
             super(MultiHead, self).__init__()
             self.L = L  # length of sequence
             self.D = D  # embedding dim
             self.H = H  # number of heads
             #---------------------------
             self.W_q = Linear(self.D, self.D, bias=False)
             self.W_k = Linear(self.D, self.D, bias=False)
             self.W_v = Linear(self.D, self.D, bias=False)
             self.W_o = Linear(self.D, self.D, bias=False)

         def forward(self, x):
             # multi-head (linear projections)
             q = self.W_q(x).view(-1, self.L, self.H, int(self.D/self.H))
             q = self.W_q(x).view(-1, self.L, self.H, int(self.D/self.H))
             k = self.W_k(x).view(-1, self.L, self.H, int(self.D/self.H))
             v = self.W_v(x).view(-1, self.L, self.H, int(self.D/self.H))
             # transpose to (H, L, D/H)
             q, k, v = q.transpose(1, 2), k.transpose(1, 2), v.transpose(1, 2)
             #-------------------
             # calculate self-attention
             k = k.transpose(2,3)
             self_attention = ((q @ k) / torch.sqrt(torch.tensor(self.D, dtype=torch.
      ↪float32))) @ v
             new_x = F.softmax(self_attention)
             #-------------------
             new_x = new_x.transpose(1, 2).contiguous()
             new_x = new_x.view(-1, self.L, self.D)
             #-------------------
             # output linear projections
             return self.W_o(new_x)
```

Feed Forward Network Layer

[0]:
```python
class MLP2Layer(torch.nn.Module):
    def __init__(self, input_size, hidden_size, output_size, bias=True):
        super(MLP2Layer, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.bias = bias
        self.dropout = Dropout(0.1)
        #---------------------------
        self.hidden = Linear(in_features=self.input_size, out_features=self.
↪hidden_size, bias=self.bias)
        self.output = Linear(in_features=self.hidden_size, out_features=self.
↪output_size, bias=self.bias)

    def forward(self, x):
        x = F.relu(self.hidden(x))
        x = self.dropout(x)
        x = self.output(x)
        return x
```

Network model definition

[0]:
```python
'''
V: vocabulary size
D: dimension of embeddings
H: number of heads in multi-head
L: legth of the sequence (number of words)
B: batch size
'''
class SelfAttentionNN(Module):
    def __init__(self, D, L, H, B, idx2vec, device):
        super(SelfAttentionNN, self).__init__()
        self.L = L  # length of sequence
        self.D = D  # embedding dim
        self.H = H  # number of heads
        self.B = B  # batch size
        self.device = device
        self.idx2vec = idx2vec.to(self.device)
        #--------------------------------------------------
        # embeddings
        self.positions = torch.arange(self.L).to(self.device)
        self.pos_emb = Embedding(num_embeddings=self.L, embedding_dim=self.D)
        #-------------------
        # multi-head attention
        self.multihead = MultiHead(self.L, self.D, self.H)
        self.norm1 = LayerNorm(self.D)
        #-------------------
        # feed-forward network
        self.ffn = MLP2Layer(self.D, self.D, self.D, bias=False)
        self.norm2 = LayerNorm(self.D)
        #-------------------
        # MLP classifier layer
        self.mlp = MLP2Layer(self.D, self.D, 2)
```

```
        #-------------------------------------------------

    def forward(self, x, mask):
        #-------------------
        # get embeddings
        x = self.idx2vec[x]
#        pdb.set_trace() # breakpoint
        # sum with positional embeddings
        x = x + self.pos_emb(self.positions)
        #-------------------
        # multi-head attention
        residual = x.clone()
        x = self.multihead(x)
        # add & norm
        x = x + residual
        x = self.norm1(x)
        #-------------------
        # feed forward
        residual = x.clone()
        x = self.ffn(x)
        # add & norm
        x = x + residual
        x = self.norm1(x)
        #-------------------
        # masked mean
        x = x * mask.reshape(-1, self.L, 1)
        #seq_len = torch.nonzero(mask).size(0)
        seq_len = self.L - (mask == 0).sum(dim=1)
        seq_len = seq_len.reshape(-1,1)
        x = (torch.sum(x, dim=1) / seq_len)
        #-------------------
        # final mlp
        x = self.mlp(x)
        return x
```

1.8 Número de parâmetros do modelo

[17]:
```
model = SelfAttentionNN(D, L, H, B, embeddings, device)
sum([torch.tensor(x.size()).prod() for x in model.parameters() if x.requires_grad]) #␣
↪trainable parameters
```

[17]:
```
tensor(692102)
```

1.9 Testando o modelo com um batch

[18]:
```
model = SelfAttentionNN(D, L_DEBUG, H, B_DEBUG, embeddings, device)
model = model.to(device)
print('Saída do modelo:')
batch_token_ids, batch_mask = batch_token_ids.to(device), batch_mask.to(device)
print(model(batch_token_ids, batch_mask))
```

```
Saída do modelo:
```

```
tensor([[ 0.1734,  0.0273],
        [-0.0888,  0.1318]], device='cuda:0', grad_fn=<AddmmBackward>)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:25: UserWarning:
Implicit dimension choice for softmax has been deprecated. Change the call to
include dim=X as an argument.
```

Definindo as funções de treino e validação.

[0]:
```
def train(dataloader, model, optimizer, criterion):

    model.train()
    loss_sum = 0.0
    for iteration, (X_batch, y_batch, mask_batch) in enumerate(dataloader):

        X_batch = X_batch.to(device)
        y_batch = y_batch.to(device)
        mask_batch = mask_batch.to(device)

        # Precisamos zerar os gradientes acumulados na iteração anterior
        optimizer.zero_grad()

        #logits = model(token_ids=X_batch)
        logits = model(X_batch, mask_batch)

        loss = criterion(logits, y_batch)
        loss_sum += loss
        # Aqui que rodamos o backpropagation para calcular os gradientes.
        loss.backward()

        # Aqui os pesos da rede são ajustados com base nos gradientes calculados
        # acima e o optimizador atualiza suas variáveis internas (taxa de
        # aprendizado, decaimento, etc).
        optimizer.step()

    average_loss = loss_sum / len(dataloader)
    return average_loss.item()
```

[0]:
```
def evaluate(dataloader, model):
    matches = 0.0
    model.eval()
    with torch.no_grad():
        for X_batch, y_batch, mask_batch in dataloader:
            X_batch = X_batch.to(device)
            y_batch = y_batch.to(device)
            mask_batch = mask_batch.to(device)

            #logits = model(token_ids=X_batch)
            logits = model(X_batch, mask_batch)
            class_predictons = logits.argmax(dim=1)

            matches += (class_predictons == y_batch).sum()

    accuracy = matches / len(dataloader.dataset)
    return accuracy.item()
```

1.10 Overfit em um batch

Antes de treinar o modelo no dataset todo, faremos overfit do modelo em um único minibatch de treino para verificar se loss vai para próximo de 0. Isso serve para depurar se a implementação do modelo está correta.

Podemos também medir se a acurácia neste minibatch chega perto de 100%. Isso serve para depurar se nossa função que mede a acurácia está correta.

Nota: se treinarmos por muitas épocas (ex: 500) é possivel que a loss vá para zero mesmo com bugs na implementação. O ideal é que a loss chege próxima a zero antes de 100 épocas.

[21]:
```
N_EPOCHS = 100

model = SelfAttentionNN(D, L, H, B, embeddings, device)
model.to(device)
optimizer = optim.Adam(model.parameters())
criterion = CrossEntropyLoss()

mydataset_debug = MyDataset(
    texts=x_train[:10],
    labels=y_train[:10],
    vocab=vocab,
    #pad_token_id=vocab['[PAD]'],
    seq_length=200)

dataloader_debug = DataLoader(mydataset_debug, batch_size=10, shuffle=True,
                              num_workers=0)

for epoch in range(N_EPOCHS):
    average_loss = train(dataloader=dataloader_debug,
                         model=model,
                         optimizer=optimizer,
                         criterion=criterion)

    train_accuracy = evaluate(dataloader=dataloader_debug, model=model)
    print(f'epoch: {epoch} '
          f'average training loss: {average_loss:.3f} '
          f'training accuracy: {train_accuracy:.3f}')
```

```
epoch: 0 average training loss: 0.701 training accuracy: 0.600
epoch: 1 average training loss: 0.677 training accuracy: 0.600
epoch: 2 average training loss: 0.662 training accuracy: 0.600
epoch: 3 average training loss: 0.655 training accuracy: 0.600
epoch: 4 average training loss: 0.640 training accuracy: 0.600
epoch: 5 average training loss: 0.618 training accuracy: 0.600
epoch: 6 average training loss: 0.609 training accuracy: 0.600
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:25: UserWarning:
Implicit dimension choice for softmax has been deprecated. Change the call to
include dim=X as an argument.
```

```
epoch: 7 average training loss: 0.560 training accuracy: 0.900
epoch: 8 average training loss: 0.550 training accuracy: 0.700
epoch: 9 average training loss: 0.470 training accuracy: 1.000
epoch: 10 average training loss: 0.393 training accuracy: 1.000
```

```
epoch: 11 average training loss: 0.281 training accuracy: 1.000
epoch: 12 average training loss: 0.196 training accuracy: 1.000
epoch: 13 average training loss: 0.112 training accuracy: 1.000
epoch: 14 average training loss: 0.045 training accuracy: 1.000
epoch: 15 average training loss: 0.017 training accuracy: 1.000
epoch: 16 average training loss: 0.006 training accuracy: 1.000
epoch: 17 average training loss: 0.003 training accuracy: 1.000
epoch: 18 average training loss: 0.001 training accuracy: 1.000
epoch: 19 average training loss: 0.000 training accuracy: 1.000
epoch: 20 average training loss: 0.000 training accuracy: 1.000
epoch: 21 average training loss: 0.000 training accuracy: 1.000
epoch: 22 average training loss: 0.000 training accuracy: 1.000
epoch: 23 average training loss: 0.000 training accuracy: 1.000
epoch: 24 average training loss: 0.000 training accuracy: 1.000
epoch: 25 average training loss: 0.000 training accuracy: 1.000
epoch: 26 average training loss: 0.000 training accuracy: 1.000
epoch: 27 average training loss: 0.000 training accuracy: 1.000
epoch: 28 average training loss: 0.000 training accuracy: 1.000
epoch: 29 average training loss: 0.000 training accuracy: 1.000
epoch: 30 average training loss: 0.000 training accuracy: 1.000
epoch: 31 average training loss: 0.000 training accuracy: 1.000
epoch: 32 average training loss: 0.000 training accuracy: 1.000
epoch: 33 average training loss: 0.000 training accuracy: 1.000
epoch: 34 average training loss: 0.000 training accuracy: 1.000
epoch: 35 average training loss: 0.000 training accuracy: 1.000
epoch: 36 average training loss: 0.000 training accuracy: 1.000
epoch: 37 average training loss: 0.000 training accuracy: 1.000
epoch: 38 average training loss: 0.000 training accuracy: 1.000
epoch: 39 average training loss: 0.000 training accuracy: 1.000
epoch: 40 average training loss: 0.000 training accuracy: 1.000
epoch: 41 average training loss: 0.000 training accuracy: 1.000
epoch: 42 average training loss: 0.000 training accuracy: 1.000
epoch: 43 average training loss: 0.000 training accuracy: 1.000
epoch: 44 average training loss: 0.000 training accuracy: 1.000
epoch: 45 average training loss: 0.000 training accuracy: 1.000
epoch: 46 average training loss: 0.000 training accuracy: 1.000
epoch: 47 average training loss: 0.000 training accuracy: 1.000
epoch: 48 average training loss: 0.000 training accuracy: 1.000
epoch: 49 average training loss: 0.000 training accuracy: 1.000
epoch: 50 average training loss: 0.000 training accuracy: 1.000
epoch: 51 average training loss: 0.000 training accuracy: 1.000
epoch: 52 average training loss: 0.000 training accuracy: 1.000
epoch: 53 average training loss: 0.000 training accuracy: 1.000
epoch: 54 average training loss: 0.000 training accuracy: 1.000
epoch: 55 average training loss: 0.000 training accuracy: 1.000
epoch: 56 average training loss: 0.000 training accuracy: 1.000
epoch: 57 average training loss: 0.000 training accuracy: 1.000
epoch: 58 average training loss: 0.000 training accuracy: 1.000
epoch: 59 average training loss: 0.000 training accuracy: 1.000
epoch: 60 average training loss: 0.000 training accuracy: 1.000
epoch: 61 average training loss: 0.000 training accuracy: 1.000
epoch: 62 average training loss: 0.000 training accuracy: 1.000
epoch: 63 average training loss: 0.000 training accuracy: 1.000
epoch: 64 average training loss: 0.000 training accuracy: 1.000
```

```
epoch: 65 average training loss: 0.000 training accuracy: 1.000
epoch: 66 average training loss: 0.000 training accuracy: 1.000
epoch: 67 average training loss: 0.000 training accuracy: 1.000
epoch: 68 average training loss: 0.000 training accuracy: 1.000
epoch: 69 average training loss: 0.000 training accuracy: 1.000
epoch: 70 average training loss: 0.000 training accuracy: 1.000
epoch: 71 average training loss: 0.000 training accuracy: 1.000
epoch: 72 average training loss: 0.000 training accuracy: 1.000
epoch: 73 average training loss: 0.000 training accuracy: 1.000
epoch: 74 average training loss: 0.000 training accuracy: 1.000
epoch: 75 average training loss: 0.000 training accuracy: 1.000
epoch: 76 average training loss: 0.000 training accuracy: 1.000
epoch: 77 average training loss: 0.000 training accuracy: 1.000
epoch: 78 average training loss: 0.000 training accuracy: 1.000
epoch: 79 average training loss: 0.000 training accuracy: 1.000
epoch: 80 average training loss: 0.000 training accuracy: 1.000
epoch: 81 average training loss: 0.000 training accuracy: 1.000
epoch: 82 average training loss: 0.000 training accuracy: 1.000
epoch: 83 average training loss: 0.000 training accuracy: 1.000
epoch: 84 average training loss: 0.000 training accuracy: 1.000
epoch: 85 average training loss: 0.000 training accuracy: 1.000
epoch: 86 average training loss: 0.000 training accuracy: 1.000
epoch: 87 average training loss: 0.000 training accuracy: 1.000
epoch: 88 average training loss: 0.000 training accuracy: 1.000
epoch: 89 average training loss: 0.000 training accuracy: 1.000
epoch: 90 average training loss: 0.000 training accuracy: 1.000
epoch: 91 average training loss: 0.000 training accuracy: 1.000
epoch: 92 average training loss: 0.000 training accuracy: 1.000
epoch: 93 average training loss: 0.000 training accuracy: 1.000
epoch: 94 average training loss: 0.000 training accuracy: 1.000
epoch: 95 average training loss: 0.000 training accuracy: 1.000
epoch: 96 average training loss: 0.000 training accuracy: 1.000
epoch: 97 average training loss: 0.000 training accuracy: 1.000
epoch: 98 average training loss: 0.000 training accuracy: 1.000
epoch: 99 average training loss: 0.000 training accuracy: 1.000
```

1.11 Treinamento e Validação no dataset todo

[22]:
```
import time
N_EPOCHS = 8

model = SelfAttentionNN(D, L, H, B, embeddings, device)
model.to(device)
#optimizer = optim.Adam(model.parameters(), lr=1e-3)
optimizer = optim.Adam(model.parameters())
criterion = CrossEntropyLoss()

ds_train = MyDataset(x_train, y_train, vocab, seq_length=200)

dl_train = torch.utils.data.DataLoader(
            dataset=ds_train,
            drop_last = False,
            shuffle = True,
```

```
            batch_size = BATCH_SIZE)

ds_valid = MyDataset(x_dev, y_dev, vocab, seq_length=200)

dl_valid =  torch.utils.data.DataLoader(
            dataset = ds_valid,
            drop_last = False,
            shuffle = False,
            batch_size = BATCH_SIZE)

for epoch in range(N_EPOCHS):
    start_time = time.time()
    train_loss = train(dataloader=dl_train,
                        model=model,
                        optimizer=optimizer,
                        criterion=criterion)
    train_time = time.time() - start_time

    start_time = time.time()
    train_accuracy = evaluate(dataloader=dl_train, model=model)
    dev_accuracy = evaluate(dataloader=dl_valid, model=model)
    eval_time = time.time() - start_time

    print(f'epoch: {epoch} '
          f'training loss: {train_loss:.3f} '
          f'training accuracy: {train_accuracy:.3f} '
          f'dev accuracy: {dev_accuracy:.3f}')

    train_examples_per_sec = len(dl_train.dataset) / train_time
    eval_examples_per_sec = (
        len(dl_train.dataset) + len(dl_valid.dataset))/ eval_time
    print(f'total training time: {train_time:.3f} '
          f'total eval time: {eval_time:.3f}')
    print(f'training examples/sec: {train_examples_per_sec:.2f} '
          f'eval examples/sec: {eval_examples_per_sec:.2f}')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:25: UserWarning:
Implicit dimension choice for softmax has been deprecated. Change the call to
include dim=X as an argument.
```

```
epoch: 0 training loss: 0.596 training accuracy: 0.752 dev accuracy: 0.755
total training time: 4.923 total eval time: 2.232
training examples/sec: 4062.55 eval examples/sec: 11199.71
epoch: 1 training loss: 0.453 training accuracy: 0.785 dev accuracy: 0.765
total training time: 4.924 total eval time: 2.235
training examples/sec: 4062.15 eval examples/sec: 11184.88
epoch: 2 training loss: 0.410 training accuracy: 0.833 dev accuracy: 0.823
total training time: 4.927 total eval time: 2.242
training examples/sec: 4059.45 eval examples/sec: 11149.18
epoch: 3 training loss: 0.386 training accuracy: 0.849 dev accuracy: 0.832
total training time: 4.932 total eval time: 2.245
training examples/sec: 4054.95 eval examples/sec: 11134.21
epoch: 4 training loss: 0.378 training accuracy: 0.836 dev accuracy: 0.820
total training time: 4.934 total eval time: 2.249
```

```
training examples/sec: 4053.75 eval examples/sec: 11118.05
epoch: 5 training loss: 0.377 training accuracy: 0.821 dev accuracy: 0.795
total training time: 4.942 total eval time: 2.254
training examples/sec: 4047.32 eval examples/sec: 11091.02
epoch: 6 training loss: 0.361 training accuracy: 0.861 dev accuracy: 0.837
total training time: 4.946 total eval time: 2.260
training examples/sec: 4043.31 eval examples/sec: 11059.89
epoch: 7 training loss: 0.345 training accuracy: 0.855 dev accuracy: 0.825
total training time: 4.952 total eval time: 2.270
training examples/sec: 4038.93 eval examples/sec: 11015.33
```

1.12 Após treinado, avaliamos o modelo no dataset de test.

É importante que essa avaliação seja feita poucas vezes para evitar o overfit no dataset de teste.

[23]:
```
ds_test  = MyDataset(x_test, y_test,  vocab, seq_length=200)

dl_test =  torch.utils.data.DataLoader(
             dataset = ds_test,
             drop_last = False,
             shuffle = False,
             batch_size = BATCH_SIZE)

test_accuracy = evaluate(dataloader=dl_test, model=model)
print(f'test accuracy: {test_accuracy:.3f}')
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:25: UserWarning:
Implicit dimension choice for softmax has been deprecated. Change the call to
include dim=X as an argument.
```

```
test accuracy: 0.821
```

1.13 End of Notebook