

✓ Notebook Processo Seletivo Aluno Especial IA-024 1S2024 FEEC-UNICAMP

versão 5 de fevereiro de 2024, 19h

versão 8 de fevereiro de 2024, 20h - ajuste variável vocab_size

Aluno: Marcus Vinícius Borela de Castro

Observações:

1. com a indisponibilidade da gpu T4 para mim no google colab após as primeiras execuções, optei por utilizar uma GPU 3090 24 gb a que tenho acesso. Então, onde se lê T4 no exercício, deve ser considerada uma GPU 3090 24gb
2. como sugerido, utilizamos diálogos com o gpt4 para apoiar nas implementação dos exercícios e nas questões teóricas.
3. optei por fazer o relatório a partir do caderno jupyter para demonstrar o passo a passo no código do impacto das questões. Separei por legibilidade as perguntas em seções. Assim, no colab, consegue-se acesso direto a partir do índice. E ficou uma documentação viva. Fora que gerar o relatório foi só imprimir como pdf o caderno.

✓ Importações

```
1 import torch
2 from torch.utils.data import Dataset, DataLoader
3 from torchtext.datasets import IMDB
4 from collections import Counter
5 import torch.nn as nn
6 import torch.optim as optim
```

```
1 import time
2 import re
3 import string
4 import math
5 import numpy as np
6 import matplotlib.pyplot as plt
7 from torch.nn.functional import one_hot
8 from torch.utils.data import DataLoader, Subset
9
```

✓ Caderno Original - execução inicial

✓ I - Vocabulário e Tokenização

```
1 %%time
2 # limit the vocabulary size to 20000 most frequent tokens
3 vocab_size = 20000
4
5 counter = Counter()
6 for (label, line) in list(IMDB(split='train')):
7     counter.update(line.split())
8
9 # create a vocabulary of the 20000 most frequent tokens
10 most_frequent_words = sorted(counter, key=counter.get, reverse=True)[:vocab_size]
11 vocab = {word: i for i, word in enumerate(most_frequent_words, 1)} # words indexed from 1 to 20000
12 vocab_size = len(vocab)
```

```
CPU times: user 8.55 s, sys: 321 ms, total: 8.87 s
Wall time: 1min 9s
```

```
1 def encode_sentence(sentence, vocab):
2     return [vocab.get(word, 0) for word in sentence.split()] # 0 for OOV
3
4 encode_sentence("I like Pizza", vocab)

[8, 35, 0]
```

- II - Dataset

```

1 %!time
2 from torch.nn.functional import one_hot
3 # Dataset Class with One-hot Encoding
4 class IMDBDataset(Dataset):
5     def __init__(self, split, vocab):
6         self.data = list(IMDB(split=split))
7         self.vocab = vocab
8
9     def __len__(self):
10         return len(self.data)
11
12     def __getitem__(self, idx):
13         label, line = self.data[idx]
14         label = 1 if label == 1 else 0
15
16         # one-hot encoding
17         X = torch.zeros(len(self.vocab) + 1)
18         for word in encode_sentence(line, self.vocab):
19             X[word] = 1
20
21         return X, torch.tensor(label)
22
23
24 # Load Data with One-hot Encoding
25 train_data = IMDBDataset('train', vocab)
26 test_data = IMDBDataset('test', vocab)
27

```

CPU times: user 691 ms, sys: 39.8 ms, total: 730 ms
Wall time: 730 ms

Ⅲ - Data Loader

[illegible]

- ✓ IV - Modelo

```

1 class OneHotMLP(nn.Module):
2     def __init__(self, vocab_size):
3         super(OneHotMLP, self).__init__()
4
5         self.fc1 = nn.Linear(vocab_size+1, 200)
6         self.fc2 = nn.Linear(200, 1)
7
8         self.relu = nn.ReLU()
9
10    def forward(self, x):
11        o = self.fc1(x.float())
12        o = self.relu(o)
13        return self.fc2(o)
14
15

```

✓ V - Laço de Treinamento - Otimização da função de Perda pelo Gradiente descendente

Observação: com a indisponibilidade do T4 para mim no google colab, utilizei uma GPU 3090 a que tenho acesso.

```

1 # Verifica se há uma GPU disponível e define o dispositivo para GPU se possível, caso contrário, usa a CPU
2 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
3 if device.type == 'cuda':
4     print('GPU:', torch.cuda.get_device_name(torch.cuda.current_device()))
5 else:
6     print('using CPU')

GPU: NVIDIA GeForce RTX 3090

1 # Model instantiation
2 model = OneHotMLP(vocab_size)

1 %%time
2
3 model = model.to(device)
4 # Define loss and optimizer
5 criterion = nn.BCEWithLogitsLoss()
6 optimizer = optim.SGD(model.parameters(), lr=0.001)
7
8 # Training loop
9 num_epochs = 5
10 for epoch in range(num_epochs):
11     start_time = time.time() # Start time of the epoch
12     model.train()
13     for inputs, labels in train_loader:
14         inputs = inputs.to(device)
15         labels = labels.to(device)
16         # Forward pass
17         outputs = model(inputs)
18         loss = criterion(outputs.squeeze(), labels.float())
19         # Backward and optimize
20         optimizer.zero_grad()
21         loss.backward()
22         optimizer.step()
23
24     end_time = time.time() # End time of the epoch
25     epoch_duration = end_time - start_time # Duration of epoch
26
27     print(f'Epoch [{epoch+1}/{num_epochs}], \
28           Loss: {loss.item():.4f}, \
29           Elapsed Time: {epoch_duration:.2f} sec')

Epoch [1/5],          Loss: 0.6924,          Elapsed Time: 20.38 sec
Epoch [2/5],          Loss: 0.6921,          Elapsed Time: 20.17 sec
Epoch [3/5],          Loss: 0.6854,          Elapsed Time: 19.73 sec
Epoch [4/5],          Loss: 0.6915,          Elapsed Time: 20.00 sec
Epoch [5/5],          Loss: 0.6954,          Elapsed Time: 20.08 sec
CPU times: user 1min 40s, sys: 193 ms, total: 1min 40s
Wall time: 1min 40s

```

✓ VI - Avaliação

```

1 %%time
2 ## evaluation
3 model.eval()
4
5 with torch.no_grad():
6     correct = 0
7     total = 0
8     for inputs, labels in test_loader:
9         inputs = inputs.to(device)
10        labels = labels.to(device)
11        outputs = model(inputs)
12        predicted = torch.round(torch.sigmoid(outputs.squeeze()))
13        total += labels.size(0)
14        correct += (predicted == labels).sum().item()
15
16 print(f'Test Accuracy: {100 * correct / total}%')

```

Test Accuracy: 55.104%
CPU times: user 19.7 s, sys: 19.9 ms, total: 19.7 s
Wall time: 19.7 s

✓ Exercícios

✓ I - Vocabulário e tokenização

✓ I.1. Alguns números

Na célula de calcular o vocabulário, aproveite o laço sobre IMDB de treinamento e utilize um segundo contador para

Calcular o número de amostras positivas e amostras negativas.

Calcule também o comprimento médio do texto em número de palavras dos textos das amostras.

```

1 %%time
2 # limit the vocabulary size to 20000 most frequent tokens
3 vocab_size = 20000
4
5 counter = Counter()
6 count_positivo = 0
7 count_negativo = 0
8 tamanho_total = 0
9 for (label, line) in list(IMDB(split='train')):
10     counter.update(line.split())
11     if label == 1:
12         count_positivo += 1
13     else:
14         count_negativo += 1
15     # print(f'tamanho: {len(line)} de {line}' )
16     tamanho_total += len(line.split())
17
18 print(f'Total de amostras: {count_negativo+count_positivo}')
19 print(f'Amostras positivas: {count_positivo}')
20 print(f'Amostras negativas: {count_negativo}')
21 print(f'Tamanho médio do texto: {tamanho_total/(count_negativo+count_positivo)}')
22
23 # create a vocabulary of the 20000 most frequent tokens
24 most_frequent_words = sorted(counter, key=counter.get, reverse=True)[:vocab_size]
25 vocab = {word: i for i, word in enumerate(most_frequent_words, 1)} # words indexed from 1 to 20000
26 vocab_size = len(vocab)
27 print(f'Tamanho vocabulário: {len(vocab)}')

```

Total de amostras: 25000
Amostras positivas: 12500
Amostras negativas: 12500
Tamanho médio do texto: 233.7872
Tamanho vocabulário: 20000
CPU times: user 1.33 s, sys: 36.1 ms, total: 1.37 s
Wall time: 1.37 s

✓ I.2. Entendo o tokenizador

As linhas 9 e 10 da célula do vocabulário são linhas típicas de programação python em listas com dicionários com laços na forma compreensão de listas ou list comprehension em inglês. Procure analisar e estudar profundamente o uso de lista e dicionário do python. Estude também a função `encode_sentence`.

**Enunciado do exercício: **

Mostre as cinco palavras mais frequentes do vocabulário e as cinco palavras menos frequentes.

Qual é o código do token que está sendo utilizado quando a palavra não está no vocabulário?

Calcule quantos tokens das frases do conjunto de treinamento que não estão no vocabulário.

✓ **I.2.a** Mostre as cinco palavras mais frequentes do vocabulário e as cinco palavras menos frequentes.

```
1 # Cinco mais e menos frequentes
2 print(f'Cinco palavras mais frequentes: {counter.most_common(5)}')
3 print(f'Cinco palavras menos frequentes: {counter.most_common()[::-6:-1]}')

Cinco palavras mais frequentes: [('the', 287032), ('a', 155096), ('and', 152664), ('of', 142972), ('to', 132568)]
Cinco palavras menos frequentes: [('Crocker', 1), ('McKenzie(Barry', 1), ('shearer', 1), ('grossest', 1), ('unemployed...', 1)]

1 def encode_sentence(sentence, vocab):
2     return [vocab.get(word, 0) for word in sentence.split()] # 0 for OOV
3
4 encode_sentence("I like Pizza", vocab)

[8, 35, 0]
```

I.2.b Qual é o código do token que está sendo utilizado quando a palavra não está no vocabulário?

R.:0 (Out Of Vocabulary): assumido no comando get se word não estiver no vocab

✓ **I.2.c** Calcule quantos tokens das frases do conjunto de treinamento que não estão no vocabulário.

```
1 def count_unknown_tokens(vocab, list_word):
2     """
3     Imprime o número de tokens distintas presentes em list_word desconhecidas no vocab
4     e o número total de ocorrências delas
5     """
6     unknown_tokens = set() # Conjunto para armazenar palavras desconhecidas
7     total_occurrences = 0 # Contador para o número total de ocorrências
8
9     for word in list_word:
10         if word not in vocab:
11             unknown_tokens.add(word) # Adiciona a palavra desconhecida ao conjunto
12             total_occurrences += 1 # Incrementa o contador
13         elif word in unknown_tokens:
14             # total_occurrences += 1 # Incrementa o contador
15
16     print(f"Número de tokens desconhecidas: {len(unknown_tokens)}")
17     print(f"Número total de ocorrências de tokens desconhecidas: {total_occurrences}")
18
1 # Exemplo de uso:
2 list_word_teste = ["maçã", "banana", "laranja", "uva", "abacaxi", "pera", "kiwi", "pêssego", "kiwi", 'pera']
3 vocab_teste = {"maçã": 1, "banana": 2, "laranja": 3, "uva": 4, "abacaxi": 5}
4
5 count_unknown_tokens(vocab_teste, list_word_teste)

Número de tokens desconhecidas: 3
Número total de ocorrências de tokens desconhecidas: 5

1 list_treino = list(IMDB(split='train'))

1 list_treino[:2]

[(1,
  'I rented I AM CURIOUS-YELLOW from my video store because of all the controversy that surrounded it when it was first released in 1967. I also heard that at first it was seized by U.S. customs if it ever tried to enter this country, therefore being a fan of films considered "controversial" I really had to see this for myself.<br /><br />The plot is centered around a young Swedish drama student named Lena who wants to learn everything she can about life. In particular she wants to focus her attentions to making some sort of documentary on what the average Swede thought about certain political issues such as the Vietnam War and race issues in the United States. In between asking politicians and ordinary denizens of Stockholm about their opinions on politics, she has sex with her drama teacher, classmates, and married men.<br /><br />What kills me about I AM CURIOUS-YELLOW is that 40 years ago, this was considered pornographic. Really, the sex and nudity scenes are few and far between, even then it\'s not shot like some cheaply made porno. While my countrymen mind find it shocking, in reality sex and nudity are a major staple in Swedish cinema. Even Ingmar Bergman, arguably their answer to good old boy John Ford, had sex scenes in his films.<br /><br />I do commend the filmmakers for the fact that any sex shown in the film is shown for artistic purposes rather than just to shock people and make money to be shown in pornographic theaters in America. I AM CURIOUS-YELLOW is a good film for anyone wanting to study the meat and potatoes (no pun intended) of Swedish cinema. But really, this film doesn\'t have much of a plot.'],
  (1,
```

"I Am Curious: Yellow" is a risible and pretentious steaming pile. It doesn't matter what one's political views are because this film can hardly be taken seriously on any level. As for the claim that frontal male nudity is an automatic NC-17, that isn't true. I've seen R-rated films with male nudity. Granted, they only offer some fleeting views, but where are the R-rated films with gaping vulvas and flapping labia? Nowhere, because they don't exist. The same goes for those crappy cable shows: schlongs swinging in the breeze but not a clitoris in sight. And those pretentious indie movies like The Brown Bunny, in which we're treated to the site of Vincent Gallo's throbbing johnson, but not a trace of pink visible on Chloe Sevigny. Before crying (or implying) "double-standard" in matters of nudity, the mentally obtuse should take into account one unavoidably obvious anatomical difference between men and women: there are no genitals on display when actresses appears nude, and the same cannot be said for a man. In fact, you generally won't see female genitals in an American film in anything short of porn or explicit erotica. This alleged double-standard is less a double standard than an admittedly depressing ability to come to terms culturally with the insides of women's bodies.'])]

```
1 list_word_treino = []
2 for target, texto in list_treino:
3     # print(texto)
4     list_word_treino.extend(texto.split())
5

1 list_word_treino[:5]

['I', 'rented', 'I', 'AM', 'CURIOUS-YELLOW']

1 count_unknown_tokens(vocab, list_word_treino)

Número de tokens desconhecidas: 260617
Número total de ocorrências de tokens desconhecidas: 566141
```

✓ 13 Reduzindo o número de amostras para 200

Uma forma simples de reduzir o número de amostras é utilizar o fatiamento de listas para selecionar apenas as primeiras 200 amostras utilizando [:200] na lista do IMDB: list(IMDB(split='train'))[:200]. Faça isto, tanto na linha 5 da célula de calcular o vocabulário como na linha 5 da célula do "II - Dataset". Com estas duas modificações, execute o notebook por completo novamente. Você verá que o tempo de processamento cairá drasticamente, para aproximadamente 1 a 2 segundos por época. Porém você vai notar que a Acurácia calculada na célula VI - Avaliação sobe para 100% ou próximo disso. Consegue justificar a razão deste resultado inesperado, entendendo que no treinamento, as perdas em cada época continuam próximas de valores com todo o dataset? Para ver a resposta, verifique agora no dataset com 200 amostras, quantas são as amostras positivas e quantas são as amostras negativas no dataset de teste.

```
1 %%time
2 # limit the vocabulary size to 20000 most frequent tokens
3 vocab_size = 20000
4
5 counter = Counter()
6 count_positivo = 0
7 count_negativo = 0
8 tamanho_total = 0
9 for (label, line) in list(IMDB(split='train'))[:200]:
10     counter.update(line.split())
11     if label == 1:
12         count_positivo += 1
13     else:
14         count_negativo += 1
15     # print(f'tamanho: {len(line)} de {line}')
16     tamanho_total += len(line.split())
17
18 print(f'Total de amostras: {count_negativo+count_positivo}')
19 print(f'Amostras positivas: {count_positivo}')
20 print(f'Amostras negativas: {count_negativo}')
21 print(f'Tamanho médio do texto: {tamanho_total/(count_negativo+count_positivo)}')
22
23 # create a vocabulary of the 20000 most frequent tokens
24 most_frequent_words = sorted(counter, key=counter.get, reverse=True)[:vocab_size]
25 vocab_lim_200 = {word: i for i, word in enumerate(most_frequent_words, 1)} # words indexed from 1 to 20000
26 vocab_size_lim_200 = len(vocab_lim_200)
27 print(f'Tamanho vocabulário limitado 200: {len(vocab_lim_200)}')

Total de amostras: 200
Amostras positivas: 200
Amostras negativas: 0
Tamanho médio do texto: 225.85
Tamanho vocabulário limitado 200: 10118
CPU times: user 383 ms, sys: 12 ms, total: 395 ms
Wall time: 394 ms
```



```

1 # Model instantiation
2 model_lim_200 = OneHotMLP(vocab_size_lim_200)

1 %%time
2 model_lim_200 = model_lim_200.to(device)
3 # Define loss and optimizer
4 criterion = nn.BCEWithLogitsLoss()
5 optimizer = optim.SGD(model_lim_200.parameters(), lr=0.001)
6
7 # Training loop
8 num_epochs = 5
9 for epoch in range(num_epochs):
10     start_time = time.time() # Start time of the epoch
11     model_lim_200.train()
12     for inputs, labels in train_loader_lim_200:
13         print(labels[0])
14         print(inputs[0])
15         inputs = inputs.to(device)
16         labels = labels.to(device)
17         # Forward pass
18         outputs = model_lim_200(inputs)
19         loss = criterion(outputs.squeeze(), labels.float())
20         # Backward and optimize
21         optimizer.zero_grad()
22         loss.backward()
23         optimizer.step()
24
25     end_time = time.time() # End time of the epoch
26     epoch_duration = end_time - start_time # Duration of epoch
27
28     print(f'Epoch [{epoch+1}/{num_epochs}], \
29           Loss: {loss.item():.4f}, \
30           Elapsed Time: {epoch_duration:.2f} sec')

tensor(1)
tensor([0., 1., 1., ..., 0., 0., 0.])
tensor(1)
tensor([0., 1., 1., ..., 0., 0., 0.])
Epoch [1/5], Loss: 0.6718, Elapsed Time: 0.19 sec
tensor(1)
tensor([0., 1., 1., ..., 0., 0., 0.])
tensor(1)
tensor([0., 1., 1., ..., 0., 0., 0.])
Epoch [2/5], Loss: 0.6707, Elapsed Time: 0.15 sec
tensor(1)
tensor([0., 1., 1., ..., 0., 0., 0.])
tensor(1)
tensor([0., 1., 1., ..., 0., 0., 0.])
Epoch [3/5], Loss: 0.6694, Elapsed Time: 0.15 sec
tensor(1)
tensor([0., 1., 1., ..., 0., 0., 0.])
tensor(1)
tensor([0., 1., 1., ..., 0., 0., 0.])
Epoch [4/5], Loss: 0.6656, Elapsed Time: 0.16 sec
tensor(1)
tensor([0., 1., 1., ..., 0., 0., 0.])
tensor(1)
tensor([0., 1., 1., ..., 0., 0., 0.])
Epoch [5/5], Loss: 0.6656, Elapsed Time: 0.15 sec
CPU times: user 775 ms, sys: 32 ms, total: 807 ms
Wall time: 806 ms

1 %%time
2 ## evaluation
3 model_lim_200.eval()
4
5 with torch.no_grad():
6     correct = 0
7     total = 0
8     for inputs, labels in test_loader_lim_200:
9         print(f' Labels: {labels[:10]}')
10        inputs = inputs.to(device)
11        labels = labels.to(device)
12        outputs = model_lim_200(inputs)
13        print(f'Outputs {outputs[:10]}')
14        predicted = torch.round(torch.sigmoid(outputs.squeeze()))
15        print(f'Predicted {predicted[:10]}')
16        total += labels.size(0)
17        correct += (predicted == labels).sum().item()
18
19 print(f'Test Accuracy: {100 * correct / total}%)

```



```

Labels: tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
Outputs tensor([[0.0474],
               [0.0446],
               [0.0451],
               [0.0119],
               [0.0381],
               [0.0663],
               [0.0773],
               [0.0578],
               [0.0571],
               [0.0649]], device='cuda:0')
Predicted tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.], device='cuda:0')
Labels: tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
Outputs tensor([[0.0533],
               [0.0315],
               [0.0833],
               [0.1085],
               [0.0474],
               [0.0586],
               [0.0791],
               [0.0546],
               [0.0657],
               [0.0540]], device='cuda:0')
Predicted tensor([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.], device='cuda:0')
Test Accuracy: 100.0%
CPU times: user 165 ms, sys: 16 ms, total: 181 ms
Wall time: 180 ms

```

- ✓ **I.3.a)** Qual é a razão pela qual o modelo preditivo conseguiu acertar 100% das amostras de teste do dataset selecionado com apenas as primeiras 200 amostras?

R.: porque os 200 primeiros registros de treinamento são todos positivos. E os primeiros 200 dados de teste também. Daí o modelo aprendeu a prever positivo. E os testes apenas testaram se era positivo.

- ✓ **I.3.b)** Modifique a forma de selecionar 200 amostras do dataset, porém garantindo que ele continue balanceado, isto é, aproximadamente 100 amostras positivas e 100 amostras negativas.

```

1 class IMDBDatasetBalanced(Dataset):
2     def __init__(self, split, vocab):
3         self.data = list(IMDB(split=split))
4         self.vocab = vocab
5
6         # Separa os exemplos 100 por classe
7         positive_examples = [item for item in self.data if item[0] == 1][:100]
8         negative_examples = [item for item in self.data if item[0] != 1][:100]
9
10        print('negative')
11        for label, input in negative_examples:
12            print(label)
13            break
14
15        print('positive')
16        for label, input in positive_examples:
17            print(label)
18            break
19
20        # Combina os exemplos das duas classes
21        self.data = positive_examples + negative_examples
22
23    def __len__(self):
24        return len(self.data)
25
26    def __getitem__(self, idx):
27        label, line = self.data[idx]
28        label = 1 if label == 1 else 0
29
30        # one-hot encoding
31        X = torch.zeros(len(self.vocab) + 1)
32        for word in encode_sentence(line, self.vocab):
33            X[word] = 1
34
35        return X, torch.tensor(label)
36 train_data_lim_200_balanced = IMDBDatasetBalanced('train', vocab_lim_200)
37 test_data_lim_200_balanced = IMDBDatasetBalanced('test', vocab_lim_200)

```

```

negative
2
positive
1

```

```

        negative
        2
        positive
        1

1 for input, label in train_data_lim_200_balanced:
2     print(label)
3     break

    tensor(1)

```

Conferindo se está balanceado:

```

1 # Crie um defaultdict com valor padrão 0 para contagem das classes
2 label_counts = {}
3
4 for _, label in train_data_lim_200_balanced:
5     if label.item() not in label_counts:
6         label_counts[label.item()] = 0
7     label_counts[label.item()] += 1
8
9 # Imprime o total por valor em label
10 for value, count in label_counts.items():
11     print(f"Label {value}: {count} exemplos")

Label 1: 100 exemplos
Label 0: 100 exemplos

```

Se o pedido fosse para balancear o dataloader, teria partido para algo parecido com o comando abaixo (não executado):

```

from torch.utils.data import WeightedRandomSampler

# Calcule os pesos para cada classe
class_counts = [100, 100] # Número de exemplos para cada classe
weights = 1.0 / torch.tensor(class_counts, dtype=torch.float)

# Crie um sampler ponderado
sampler = WeightedRandomSampler(weights, len(train_data), replacement=True)

# Crie o DataLoader com o sampler
train_loader_balanced = DataLoader(train_data, batch_size=batch_size, sampler=sampler)

```


II - Dataset

Precisamos entender como funciona a classe IMDBDataset. Ela é a classe responsável para acessar cada amostra do dataset.

Em primeiro lugar precisamos entender qual será a entrada da rede neural para decidir se o texto é uma crítica positiva ou negativa. Uma das formas mais simples de construir um modelo preditivo é com base nas palavras utilizadas no texto. A distribuição das palavras de um texto tem alta correlação com o fato do texto estar falando bem ou falando mal de um filme. Certamente é estimativa que possui seus erros, mas é a forma mais simples e eficiente de se fazer uma análise de sentimento ou de maneira geral uma classificação de um texto. Esse método é denominado "Bag of Words". A entrada da rede neural, para cada amostra, será um vetor de comprimento do vocabulário, com valores todos zero, com exceção dos tokens que aparecem no texto da amostra. Esse método de codificação é também denominado "One-Hot". Estude o código da classe IMDBDataset fazendo experimentos e perguntas ao chatGPT para entender com profundidade esta classe.

Enunciado do exercício:

II.1 Investigação inicial

-  II.1.a) Investigue o dataset criado na linha 24. Faça um código que aplique um laço sobre o dataset train_data e calcule novamente quantas amostras positivas e negativas do dataset.

```

1 %%time
2 # limit the vocabulary size to 20000 most frequent tokens
3 vocab_size = 20000
4
5 counter = Counter()
6 for (label, line) in list(IMDB(split='train')):
7     counter.update(line.split())
8
9 # create a vocabulary of the 20000 most frequent tokens
10 most_frequent_words = sorted(counter, key=counter.get, reverse=True)[:vocab_size]
11 vocab = {word: i for i, word in enumerate(most_frequent_words, 1)} # words indexed from 1 to 20000
12 vocab_size = len(vocab)

```

```

CPU times: user 1.1 s, sys: 36 ms, total: 1.14 s
Wall time: 1.14 s

```

```

1 %%time
2 from torch.nn.functional import one_hot
3 # Dataset Class with One-hot Encoding
4 class IMDBDataset(Dataset):
5     def __init__(self, split, vocab):
6         self.data = list(IMDB(split=split))
7         self.vocab = vocab
8
9     def __len__(self):
10         return len(self.data)
11
12     def __getitem__(self, idx):
13         label, line = self.data[idx]
14         label = 1 if label == 1 else 0
15
16         # one-hot encoding
17         X = torch.zeros(len(self.vocab) + 1)
18         for word in encode_sentence(line, self.vocab):
19             X[word] = 1
20
21         return X, torch.tensor(label)
22
23
24 # Load Data with One-hot Encoding
25 train_data = IMDBDataset('train', vocab)
26 test_data = IMDBDataset('test', vocab)
27

```

```

CPU times: user 678 ms, sys: 36.1 ms, total: 714 ms
Wall time: 713 ms

```

```

1 # Crie um defaultdict com valor padrão 0 para contagem das classes
2 label_counts = {}
3
4 for _, label in train_data:
5     if label.item() not in label_counts:
6         label_counts[label.item()] = 0
7     label_counts[label.item()] += 1
8
9 # Imprime o total por valor em label
10 for value, count in label_counts.items():
11     print(f"Label {value}: {count} exemplos")

```

```

Label 1: 12500 exemplos
Label 0: 12500 exemplos

```

- ✓ II.1.b) Calcule também o número médio de palavras codificadas em cada vetor one-hot. Compare este valor com o comprimento médio de cada texto (contado em palavras), conforme calculado no exercício I.1.c. e explique a diferença.

```

1 len(train_data)

25000

1 total_palavras = 0
2 for input, _ in train_data:
3     # print(input)
4     # print(torch.sum(input).item())
5     total_palavras += torch.sum(input).item()
6 print(f'A média de palavras em cada vetor é {total_palavras/len(train_data)}')

```

```

A média de palavras em cada vetor é 133.09548

```

Em I.1.c o tamanho médio era 233.78 palavras por sentença. O valor menor 133.09 é explicado por dois motivos:

1. A não contagem de repetições (trata-se de um bag of words binário, com valores 0 ou 1 conforme o token esteja na sentença, e não com quantidade de ocorrências do token, seria uma versão "count").
2. Como o vocabulário só tem 20000 tokens diferentes, uma grande parte de tokens distintas (260617, conforme exercício I.2.C) não foram contadas. Na realidade todas elas foram consideradas como uma única token (out of vocabulary) na posição 0 do tensor.

A rede neural será alimentada pelo vetor one-hot (quais suas dimensões) e fará uma predição da probabilidade do texto associado ao one-hot ser uma mensagem positiva.

✓ II.2 Aumentando a eficiência do treinamento com o uso da GPU T4

O código do notebook está preparado para executar tanto com ambiente usando CPU como com GPU, entretanto o ganho de velocidade está sendo reduzido de 45 segundos para 29 segundos que é um ganho muito aquém do esperado que seria ter um speedup entre 7 e 11 vezes dependendo da aplicação. Vamos entender a razão desta baixa eficiência e corrigir o problema.

A GPU é utilizada durante o treinamento do modelo, onde é utilizada a técnica de minimização da Loss utilizando o gradiente descendente. Isso ocorre na segunda célula do "V - Laço de Treinamento". Iremos analisar os detalhes mais à frente, para por enquanto basta entender onde a GPU é utilizada. A linha 17 é onde o modelo está fazendo a predição (passo forward), dado a entrada, calcula a saída da rede (muitas vezes chamado de logito) e o cálculo da loss está sendo feito na linha seguinte e o cálculo do gradiente ocorre na linha 21 e a linha 22 é onde ocorre o ajuste dos parâmetros (weights) da rede neural fazendo ela minimizar a Loss. Esse é o processo que é mais demorado e onde a GPU tem muitos ganhos, pois envolve praticamente apenas multiplicação de matrizes. Existem apenas 3 linhas que controlam o uso da GPU que servem para colocar o modelo, a entrada e a saída esperada (labels) na GPU: linhas 3, 14 e 15, respectivamente.

Enunciado do exercício: Com a o notebook configurado para GPU T4, meça o tempo de dois laços dentro do for da linha 13 (coloque um break após dois laços) e determine quanto demora demora para o passo de forward (linhas 14 a 18), para o backward (linhas 20, 21 e 22) e o tempo total de um laço. Faça as contas e identifique o trecho que é mais demorado.

Observação: com a indisponibilidade do T4 para mim no google colab, utilizei uma GPU 3090 a que tenho acesso.

```
1 # Verifica se há uma GPU disponível e define o dispositivo para GPU se possível, caso contrário, usa a CPU
2 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
3 if device.type == 'cuda':
4     print('GPU:', torch.cuda.get_device_name(torch.cuda.current_device()))
5 else:
6     print('using CPU')
```

GPU: NVIDIA GeForce RTX 3090

✓ II.2.a) Tempo do laço = ; Tempo do forward = ; Tempo do backward = ; Conclusão.

```
1 %%time
2 batch_size = 128
3 # define dataloaders
4 train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
5 test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False)
6
```

CPU times: user 7.79 ms, sys: 0 ns, total: 7.79 ms
Wall time: 7.76 ms

```
1 class OneHotMLP(nn.Module):
2     def __init__(self, vocab_size):
3         super(OneHotMLP, self).__init__()
4
5         self.fc1 = nn.Linear(vocab_size+1, 200)
6         self.fc2 = nn.Linear(200, 1)
7
8         self.relu = nn.ReLU()
9
10    def forward(self, x):
11        o = self.fc1(x.float())
12        o = self.relu(o)
13        return self.fc2(o)
14
15
```

```

1 # Model instantiation
2 model = OneHotMLP(vocab_size)

1 %%time
2
3 model = model.to(device)
4 # Define loss and optimizer
5 criterion = nn.BCEWithLogitsLoss()
6 optimizer = optim.SGD(model.parameters(), lr=0.001)
7
8 # Training loop
9 num_epochs = 5
10 for epoch in range(num_epochs):
11     start_time = time.time() # Start time of the epoch
12     model.train()
13     tempo_forward = 0
14     tempo_backward = 0
15     tempo_move_gpu = 0
16
17     for cnt, (inputs, labels) in enumerate(train_loader):
18         if cnt >= 2:
19             break
20         print(f"Batch {cnt+1}")
21         # mover para gpu
22         move_gpu_begin_time = time.time()
23         inputs = inputs.to(device)
24         labels = labels.to(device)
25         tempo_move_gpu += time.time() - move_gpu_begin_time
26         # Forward pass
27         forward_begin_time = time.time()
28         outputs = model(inputs)
29         loss = criterion(outputs.squeeze(), labels.float())
30         tempo_forward += time.time() - forward_begin_time
31         # Backward and optimize
32         backward_begin_time = time.time()
33         optimizer.zero_grad()
34         loss.backward()
35         optimizer.step()
36         tempo_backward += time.time() - backward_begin_time
37
38     tempo_total = time.time() - start_time
39
40
41     end_time = time.time() # End time of the epoch
42     epoch_duration = end_time - start_time # Duration of epoch
43
44     print(f'Epoch [{epoch+1}/{num_epochs}], \
45           Loss: {loss.item():.4f}, \
46           Elapsed Time: {epoch_duration:.2f} sec')
47     break
48 print(f'Tempo total = {round(tempo_total,3)}')
49 print(f'Tempo move tensor to gpu = {round(tempo_move_gpu,5)} {round(tempo_move_gpu/tempo_total,6)}%')
50 print(f'Tempo forward = {round(tempo_forward,5)} {round(tempo_forward/tempo_total,6)}%')
51 print(f'Tempo backward = {round(tempo_backward,5)} {round(tempo_backward/tempo_total,6)}%')
52

```

```

Batch 1
Batch 2
Epoch [1/5],          Loss: 0.6946,          Elapsed Time: 0.32 sec
Tempo total = 0.321
Tempo move tensor to gpu = 0.02407 0.074982%
Tempo forward = 0.00111 0.003464%
Tempo backward = 0.0014 0.004352%
CPU times: user 498 ms, sys: 20.1 ms, total: 518 ms
Wall time: 322 ms

```

R.: (soma dos 2 laços)

Tempo total=0.321; Tempo de movimentação para gpu = 0.02 (7%); Tempo do forward = 0.0011 (3%); Tempo do backward = 0.0014 (4%);

Conclusão

O tempo de percorrimento do dataloader é o maior gargalo, aproximadamente 86%.

✓ **II.2.b)** Trecho que precisa ser otimizado. (Esse é um problema mais difícil)

R.: Esse loop precisa ser otimizado! (ver mais detalhes na solução que se segue em II.2.c.)

```

1 # Dataset Class with One-hot Encoding
2 class IMDBDataset(Dataset):
3     (...)
4     def __getitem__(self, idx):
5         label, line = self.data[idx]
6         label = 1 if label == 1 else 0
7
8         # one-hot encoding
9         X = torch.zeros(len(self.vocab) + 1)
10        for word in encode_sentence(line, self.vocab):
11            X[word] = 1
12
13        return X, torch.tensor(label)
14

```

✓ **II.2.c)** Otimize o código e explique aqui.

Após esta otimização, é esperado que o tempo de processamento de cada época caia tanto para execução em CPU (da ordem de 10 segundos por época) como para GPU (da ordem de 1 a 2 segundos por época). Isso utilizando as 25 mil amostras do dataset IMDB inteiro.

Agora que a execução está bem mais otimizada em tempos de execução, mantenha o dataset completo: 25 mil amostras e vamos analisar um outro fator importante que é a escolha do LR (Learning Rate)

R.: Primeiro troquei o `encode_sentence` por `return_vocab_in_sentence` que não passa mais de uma vez em repetições de uma palavra (usado `set`).

```

1 def return_vocab_in_sentence(sentence, vocab):
2     return [vocab.get(word, 0) for word in set(sentence.split())] # 0 for OOV
3
4 encode_sentence("I like Pizza", vocab)

[35, 0, 8]

```

Depois, retirado loop

```

for word in encode_sentence(line, self.vocab):
    X[word] = 1

```

para

```

X[encode_sentence(line, self.vocab) vocab] = 1

```

E, por fim, como o conjunto de dados não é muito grande, optei por carregar os tensores no **init**

Em contrapartida, haverá economia de memória ao não mais armazenar os textos (`self.data`). Segue código otimizado.

Obs.: fora do escopo deste exercício a avaliação se haverá ou não economia de memória nessa troca. Mas, de tempo, com certeza (cache de valores calculados).

```

1 %%time
2 from torch.nn.functional import one_hot
3 # Dataset Class with One-hot Encoding
4 class IMDBDataset(Dataset):
5     def __init__(self, split, vocab):
6         data = list(IMDB(split=split))
7         self.vocab = vocab
8         self.labels = [torch.tensor(1) if item[0] == 1 else torch.tensor(0) for item in data]
9
10        self.sentences = []
11        for label, line in data:
12            X = torch.zeros(len(self.vocab) + 1)
13            X[return_vocab_in_sentence(line, self.vocab)] = 1
14            self.sentences.append(X)
15
16
17        def __len__(self):
18            return len(self.labels)
19
20        def __getitem__(self, idx):
21            # print('retornando idx', idx, self.sentences[idx], self.labels[idx])
22            return self.sentences[idx], self.labels[idx]
23
24
25 # Load Data with One-hot Encoding
26 train_data = IMDBDataset('train', vocab)
27 test_data = IMDBDataset('test', vocab)

```

```

CPU times: user 4.33 s, sys: 92.1 ms, total: 4.43 s
Wall time: 4.42 s

```

```

1 %%time
2 batch_size = 128
3 # define dataloaders
4 train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
5 test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=False)
6

```

```

CPU times: user 53.8 ms, sys: 0 ns, total: 53.8 ms
Wall time: 53.6 ms

```

```

1 %%time
2
3 model = model.to(device)
4 # Define loss and optimizer
5 criterion = nn.BCEWithLogitsLoss()
6 optimizer = optim.SGD(model.parameters(), lr=0.001)
7
8 # Training loop
9 num_epochs = 5
10 for epoch in range(num_epochs):
11     start_time = time.time() # Start time of the epoch
12     model.train()
13     tempo_forward = 0
14     tempo_backward = 0
15     tempo_move_gpu = 0
16
17     for cnt, (inputs, labels) in enumerate(train_loader):
18         if cnt >= 2:
19             break
20         print(f"Batch {cnt+1}")
21         # mover para gpu
22         move_gpu_begin_time = time.time()
23         inputs = inputs.to(device)
24         labels = labels.to(device)
25         tempo_move_gpu += time.time() - move_gpu_begin_time
26         # Forward pass
27         forward_begin_time = time.time()
28         outputs = model(inputs)
29         loss = criterion(outputs.squeeze(), labels.float())
30         tempo_forward += time.time() - forward_begin_time
31         # Backward and optimize
32         backward_begin_time = time.time()
33         optimizer.zero_grad()
34         loss.backward()
35         optimizer.step()
36         tempo_backward += time.time() - backward_begin_time
37
38     tempo_total = time.time() - start_time
39
40
41     end_time = time.time() # End time of the epoch
42     epoch_duration = end_time - start_time # Duration of epoch
43
44     print(f'Epoch [{epoch+1}/{num_epochs}], \
45           Loss: {loss.item():.4f}, \
46           Elapsed Time: {epoch_duration:.2f} sec')
47     break
48 print(f'Tempo total = {round(tempo_total,3)}')
49 print(f'Tempo move tensor to gpu = {round(tempo_move_gpu,5)} {round(tempo_move_gpu/tempo_total,6)}%')
50 print(f'Tempo forward = {round(tempo_forward,5)} {round(tempo_forward/tempo_total,6)}%')
51 print(f'Tempo backward = {round(tempo_backward,5)} {round(tempo_backward/tempo_total,6)}%')
52

```

```

Batch 1
Batch 2
Epoch [1/5],          Loss: 0.6939,          Elapsed Time: 0.03 sec
Tempo total = 0.034
Tempo move tensor to gpu = 0.02415 0.703817%
Tempo forward = 0.00087 0.025221%
Tempo backward = 0.00158 0.046106%
CPU times: user 19.1 ms, sys: 16.1 ms, total: 35.2 ms
Wall time: 34.9 ms

```

Tempo total nos 2 laços caiu de 0.321 para 0.034, para cerca de 10%.

Tempo finais (apenas para comparativo dentro do loop): Movimentação para gpu = 0.02 (70%); Tempo do forward = 0.0008 (2.5%); Tempo do backward = 0.0015 (4,6%); Percorrimento dataloader: 22,9%

✓ II.3 Escolhendo um bom valor de LR

Enunciado do exercício: Faça a melhor escolha do LR, analisando o valor da acurácia no conjunto de teste, utilizando para cada valor de LR, a acurácia obtida. Faça um gráfico de Acurácia vs LR e escolha o LR que forneça a maior acurácia possível.

Ações prévias

Definindo lista de valores de LR

Melhor usar versão de logspace que gera num_values valores igualmente espaçados em uma escala logarítmica entre $10^{\text{min_lr}}$ e $10^{\text{max_lr}}$.

```
1 num_values = 8
2 # Define the minimum and maximum learning rate and the number of values
3 min_lr = 0.000001
4 max_lr = 0.1
5 learning_rates_list = np.linspace(min_lr, max_lr, num=num_values)
6 print('linspace', learning_rates_list)
7
8 # Generate the learning rates
9 min_lr = -8 # 10^-4
10 max_lr = -1 # 10^-1
11
12 learning_rates_list = np.logspace(min_lr, max_lr, num=num_values)
13 print('logspace', learning_rates_list)
14 # Now learning_rates is an array with 20 values equally spaced between 0.0001 and 0.1

linspace [1.00000000e-06 1.42865714e-02 2.85721429e-02 4.2857143e-02
 5.71432857e-02 7.14288571e-02 8.57144286e-02 1.00000000e-01]
logspace [1.e-08 1.e-07 1.e-06 1.e-05 1.e-04 1.e-03 1.e-02 1.e-01]
```

Criando funções de treino e avaliação para chamar no loop de para cada learning rate

```
1 def evaluate(model, test_loader):
2     ## evaluation
3     model.eval()
4
5     with torch.no_grad():
6         correct = 0
7         total = 0
8         for inputs, labels in test_loader:
9             inputs = inputs.to(device)
10            labels = labels.to(device)
11            outputs = model(inputs)
12            predicted = torch.round(torch.sigmoid(outputs.squeeze()))
13            total += labels.size(0)
14            correct += (predicted == labels).sum().item()
15        test_accuracy = 100 * correct / total
16
17    return test_accuracy
```

```

1 def train_model(model, criterion, optimizer, num_epochs, train_loader, test_loader, verbose:bool=True):
2
3     # Training loop
4     for epoch in range(num_epochs):
5         epoch_start_time = time.time() # Start time of the epoch
6         model.train()
7         for cnt_batch, (inputs, labels) in enumerate(train_loader):
8
9
10            # print(f"Batch {cnt+1}")
11            # mover para gpu
12            inputs = inputs.to(device)
13            labels = labels.to(device)
14            # Forward pass
15            outputs = model(inputs)
16            if verbose and cnt_batch==0:
17                # Verifique as dimensões dos tensores
18                print("Dimensões dos outputs:", outputs.shape)
19                print("Dimensões dos labels:", labels.shape)
20            loss = criterion(outputs.squeeze(), labels.float())
21            # Backward and optimize
22            optimizer.zero_grad()
23            loss.backward()
24            optimizer.step()
25
26            epoch_duration = time.time() - epoch_start_time # Duration of epoch
27
28            if verbose:
29                print(f'Epoch [{epoch+1}/{num_epochs}], \
30                    Loss: {loss.item():.4f}, \
31                    Elapsed Time: {epoch_duration:.2f} sec')
32
33            # Evaluate the model and record the test accuracy
34            test_accuracy = evaluate(model, test_loader)
35
36            return test_accuracy
37
38

```

✓ II.3.a) Gráfico Acurácia vs LR

Efetuando o treinamento

```

1 train_loader = DataLoader(train_data,
2                           batch_size= batch_size,
3                           shuffle=True,
4                           num_workers=0,
5                           pin_memory=True)

1 print(f"train_loader tem {len(train_loader)} batches")

train_loader tem 196 batches

1 # Initialize a dictionary to record test accuracy for each learning rate
2 test_accuracies_by_lr = {}
3
4 for cnt_treino, lr in enumerate(learning_rates_list):
5     # Model instantiation
6     model = OneHotMLP(vocab_size)
7     model = model.to(device)
8     criterion = nn.BCEWithLogitsLoss()
9     optimizer = optim.SGD(model.parameters(), lr=lr)
10    train_start_time = time.time() # Start time of the epoch
11    test_accuracy = train_model(model, criterion, optimizer, num_epochs=5, train_loader, test_loader)
12
13    train_duration = time.time() - train_start_time # Duration of epoch
14
15    test_accuracies_by_lr[lr] = test_accuracy
16    print(f'Treino [{cnt_treino + 1}/{len(learning_rates_list)}]: lr={lr}; test_accuracy={round(test_accuracy,4)} (train_duration {1

Epoch [1/5],          Loss: 0.6950,          Elapsed Time: 1.53 sec
Epoch [2/5],          Loss: 0.6937,          Elapsed Time: 1.57 sec
Epoch [3/5],          Loss: 0.6930,          Elapsed Time: 1.51 sec
Epoch [4/5],          Loss: 0.6922,          Elapsed Time: 1.57 sec
Epoch [5/5],          Loss: 0.6960,          Elapsed Time: 1.58 sec
Treino [1/8]: lr=1e-08; test_accuracy=47.972 (train_duration 8.88 sec)
Epoch [1/5],          Loss: 0.6902,          Elapsed Time: 1.50 sec
Epoch [2/5],          Loss: 0.6940,          Elapsed Time: 1.53 sec
Epoch [3/5],          Loss: 0.6904,          Elapsed Time: 1.53 sec

```

```

Epoch [4/5],          Loss: 0.6970,          Elapsed Time: 1.58 sec
Epoch [5/5],          Loss: 0.6929,          Elapsed Time: 1.56 sec
Treino [2/8]: lr=1e-07; test_accuracy=50.336 (train_duration 8.83 sec)
Epoch [1/5],          Loss: 0.6848,          Elapsed Time: 1.53 sec
Epoch [2/5],          Loss: 0.6880,          Elapsed Time: 1.50 sec
Epoch [3/5],          Loss: 0.6896,          Elapsed Time: 1.56 sec
Epoch [4/5],          Loss: 0.7040,          Elapsed Time: 1.56 sec
Epoch [5/5],          Loss: 0.6877,          Elapsed Time: 1.57 sec
Treino [3/8]: lr=1e-06; test_accuracy=49.98 (train_duration 8.84 sec)
Epoch [1/5],          Loss: 0.6927,          Elapsed Time: 1.55 sec
Epoch [2/5],          Loss: 0.6885,          Elapsed Time: 1.57 sec
Epoch [3/5],          Loss: 0.6840,          Elapsed Time: 1.54 sec
Epoch [4/5],          Loss: 0.6845,          Elapsed Time: 1.53 sec
Epoch [5/5],          Loss: 0.6976,          Elapsed Time: 1.57 sec
Treino [4/8]: lr=1e-05; test_accuracy=50.0 (train_duration 8.89 sec)
Epoch [1/5],          Loss: 0.7043,          Elapsed Time: 1.57 sec
Epoch [2/5],          Loss: 0.6957,          Elapsed Time: 1.56 sec
Epoch [3/5],          Loss: 0.6927,          Elapsed Time: 1.55 sec
Epoch [4/5],          Loss: 0.7012,          Elapsed Time: 1.58 sec
Epoch [5/5],          Loss: 0.6941,          Elapsed Time: 1.56 sec
Treino [5/8]: lr=0.0001; test_accuracy=50.008 (train_duration 8.92 sec)
Epoch [1/5],          Loss: 0.6916,          Elapsed Time: 1.59 sec
Epoch [2/5],          Loss: 0.6984,          Elapsed Time: 1.56 sec
Epoch [3/5],          Loss: 0.6940,          Elapsed Time: 1.59 sec
Epoch [4/5],          Loss: 0.6859,          Elapsed Time: 1.58 sec
Epoch [5/5],          Loss: 0.6922,          Elapsed Time: 1.57 sec
Treino [6/8]: lr=0.001; test_accuracy=57.86 (train_duration 9.06 sec)
Epoch [1/5],          Loss: 0.6838,          Elapsed Time: 1.55 sec
Epoch [2/5],          Loss: 0.6625,          Elapsed Time: 1.62 sec
Epoch [3/5],          Loss: 0.6099,          Elapsed Time: 1.63 sec
Epoch [4/5],          Loss: 0.5408,          Elapsed Time: 1.56 sec
Epoch [5/5],          Loss: 0.5349,          Elapsed Time: 1.53 sec
Treino [7/8]: lr=0.01; test_accuracy=79.12 (train_duration 9.02 sec)
Epoch [1/5],          Loss: 0.4516,          Elapsed Time: 1.56 sec
Epoch [2/5],          Loss: 0.3425,          Elapsed Time: 1.60 sec
Epoch [3/5],          Loss: 0.2768,          Elapsed Time: 1.59 sec
Epoch [4/5],          Loss: 0.2530,          Elapsed Time: 1.59 sec
Epoch [5/5],          Loss: 0.3710,          Elapsed Time: 1.58 sec
Treino [8/8]: lr=0.1; test_accuracy=75.084 (train_duration 9.07 sec)

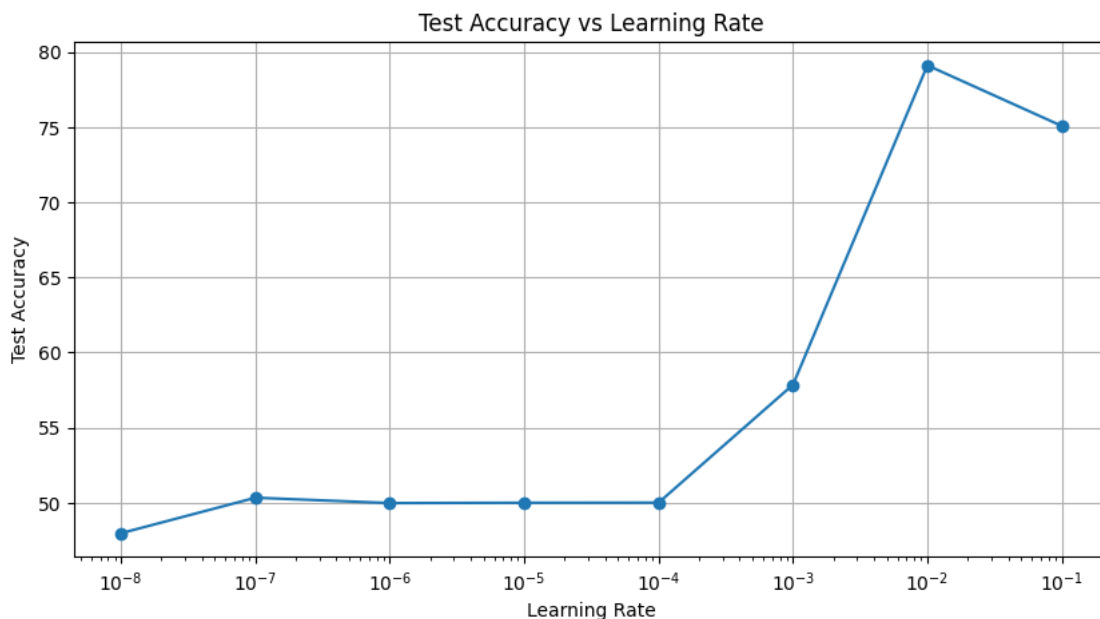
```

Gerando o gráfico

```

1
2 # Plot test accuracy for each learning rate
3 plt.figure(figsize=(10, 5))
4 plt.plot(list(test_accuracies_by_lr.keys()), list(test_accuracies_by_lr.values()), marker='o')
5 plt.xscale('log')
6 plt.xlabel('Learning Rate')
7 plt.ylabel('Test Accuracy')
8 plt.title('Test Accuracy vs Learning Rate')
9 plt.grid(True)
10 plt.show()
11

```



✓ II.3.b) Valor ótimo do LR

R.: Segue resposta

Obs.: treinamentos diferentes retornaram LR diferentes como melhores. Não é escopo deste exercício tratar a repetibilidade dos resultados.

```
1 best_lr = max(test_accuracies_by_lr, key=test_accuracies_by_lr.get)
2 print(f'The learning rate with the highest test accuracy is {best_lr}')

The learning rate with the highest test accuracy is 0.01
```

- ✓ **II.3.c)** Mostre a equação utilizada no gradiente descendente e qual é o papel do LR no ajuste dos parâmetros (weights) do modelo da rede neural.

R.:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta)$$

Nesta equação:

(θ) - são os parâmetros (ou pesos) do modelo

(η) é a taxa de aprendizado (LR).

$(\nabla_{\theta} J(\theta))$ é o gradiente da função de perda J com relação aos parâmetros (θ) .

O papel do LR no ajuste dos parâmetros do modelo da rede neural é determinar o tamanho do passo em cada iteração do gradiente descendente. Um LR alto pode fazer com que o algoritmo de otimização dê passos maiores e possivelmente pule o mínimo global. Por outro lado, um LR muito baixo pode fazer com que o algoritmo de otimização dê passos muito pequenos, o que pode resultar em um tempo de treinamento muito longo ou o algoritmo pode ficar preso em um mínimo local. Portanto, a escolha do LR é um compromisso entre a velocidade de treinamento e a capacidade do modelo de encontrar o mínimo global da função de perda.

✓ II.4 Otimizando o tokenizador

Agora que a convergência da Loss está melhor, vamos experimentar os parâmetros do tokenizador, isto é, como as palavras estão codificadas em tokens. Observe novamente o vocab criado na parte I - Vocabulário e Tokenização. Perceba como as pontuações estão influenciando nos tokens criados e como o uso de letras maiúsculas e minúsculas também podem atrapalhar a consistência dos tokenizador em representar o significado semântico das palavras. Experimente rodar o `encode_sentence` com frases que tenham pontuações e letras maiúsculas e minúsculas. Baseado nessas informações, procure melhorar a forma de tokenizar o dataset. Enunciado do exercício: Melhore a forma de tokenizar, isto é, pré-processar o dataset de modo que a codificação seja indiferente das palavras serem escritas com maiúsculas ou minúsculas e sejam pouco influenciadas pelas pontuações.

Identificando o problema

```
1 def encode_sentence(sentence, vocab):
2     return [vocab.get(word, 0) for word in sentence.split()] # 0 for OOV
3
4 encode_sentence("I like Pizza", vocab)

[8, 35, 0]

1 for sentence in ["I like Pizza", "I like pizza", "I Like pizza", "I like Pizza.", "I like Pizza .", "I like, Pizza", ]:
2     coded_sentence = encode_sentence(sentence, vocab)
3     print(f'Para sentença {sentence}, lista de palavras: {sentence.split()} retornou: {coded_sentence}')

Para sentença I like Pizza, lista de palavras: ['I', 'like', 'Pizza'] retornou: [8, 35, 0]
Para sentença I like pizza, lista de palavras: ['I', 'like', 'pizza'] retornou: [8, 35, 14772]
Para sentença I Like pizza, lista de palavras: ['I', 'Like', 'pizza'] retornou: [8, 892, 14772]
Para sentença I like Pizza., lista de palavras: ['I', 'like', 'Pizza.'] retornou: [8, 35, 0]
Para sentença I like Pizza ., lista de palavras: ['I', 'like', 'Pizza', '.'] retornou: [8, 35, 0, 422]
Para sentença I like, Pizza, lista de palavras: ['I', 'like,', 'Pizza'] retornou: [8, 2280, 0]
```

O problema está no separador de tokens. Criado novo código.

```

1 def generate_list_of_tokens(sentence):
2     """
3     Given a string sentence, returns a list of tokens.
4     It desconsiders if word is in lower or upper case.
5     It is not impacted by punctuation
6     """
7     # Convert to lowercase
8     sentence = sentence.lower()
9
10    # Remove punctuation
11    translator = str.maketrans('', '', string.punctuation)
12    sentence = sentence.translate(translator)
13
14    # Tokenize the sentence using regular expressions
15    tokens = re.findall(r'\b\w+\b', sentence)
16
17    return tokens
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671

```

II - Dataset (já com código otimizado do exercício anterior II.2.c)

```
1 %%time
2 from torch.nn.functional import one_hot
3 # Dataset Class with One-hot Encoding
4 class IMDBDataset(Dataset):
5     def __init__(self, split, vocab):
6         data = list(IMDB(split=split))
7         self.vocab = vocab
8         self.labels = [torch.tensor(1) if item[0] == 1 else torch.tensor(0) for item in data]
9
10        self.sentences = []
11        for label, line in data:
12            X = torch.zeros(len(self.vocab) + 1)
13            X[return_vocab_in_sentence(line, self.vocab)] = 1
14            self.sentences.append(X)
15
16
17        def __len__(self):
18            return len(self.labels)
19
20        def __getitem__(self, idx):
21            # print('retornando idx', idx, self.sentences[idx], self.labels[idx])
22            return self.sentences[idx], self.labels[idx]
23
24
25 # Load Data with One-hot Encoding
26 train_data = IMDBDataset('train', vocab)
27 test_data = IMDBDataset('test', vocab)
28 print(f'Tamanho de train_data {len(train_data)}')
29 print(f'Tamanho de test_data {len(test_data)}')
```

Tamanho de train_data 25000
Tamanho de test_data 25000
CPU times: user 7.2 s, sys: 40 ms, total: 7.24 s
Wall time: 7.24 s

- ✓ **II.4.b)** Recalcule novamente os valores do exercício I.2.c - número de tokens unknown, e apresente uma tabela comparando os novos valores com os valores obtidos com o tokenizador original e justifique os resultados obtidos.

```
1 %%time
2 # limit the vocabulary size to 20000 most frequent tokens
3 vocab_size = 20000
4
5 counter = Counter()
6 count_positivo = 0
7 count_negativo = 0
8 tamanho_total = 0
9 for (label, line) in list(IMDB(split='train')):
10     line_processed = generate_list_of_tokens(line)
11     counter.update(line_processed)
12     if label == 1:
13         count_positivo += 1
14     else:
15         count_negativo += 1
16     # print(f'tamanho: {len(line)} de {line}')
17     tamanho_total += len(line_processed)
18
19 print(f'Total de amostras: {count_negativo+count_positivo}')
20 print(f'Amostras positivas: {count_positivo}')
21 print(f'Amostras negativas: {count_negativo}')
22 print(f'Tamanho médio do texto: {tamanho_total/(count_negativo+count_positivo)}')
```

23

```
24 # create a vocabulary of the 20000 most frequent tokens
25 most_frequent_words = sorted(counter, key=counter.get, reverse=True)[:vocab_size]
26 vocab = {word: i for i, word in enumerate(most_frequent_words, 1)} # words indexed from 1 to 20000
27 vocab_size = len(vocab)
28 print(f'Tamanho vocabulário: {len(vocab)}')
```

29

Total de amostras: 25000
Amostras positivas: 12500
Amostras negativas: 12500
Tamanho médio do texto: 232.83608
Tamanho vocabulário: 20000
CPU times: user 2.47 s, sys: 16 ms, total: 2.49 s
Wall time: 2.48 s

```

1 # Cinco mais e menos frequentes
2 print(f'Cinco palavras mais frequentes: {counter.most_common(5)}')
3 print(f'Cinco palavras menos frequentes: {counter.most_common()[: -6: -1]}')

Cinco palavras mais frequentes: [('the', 334730), ('and', 162252), ('a', 161958), ('of', 145326), ('to', 135046)]
Cinco palavras menos frequentes: [('mckenziebarry', 1), ('grossest', 1), ('dunny', 1), ('fitzgibbon', 1), ('snacka', 1)]

```

Antes

```

Cinco palavras mais frequentes: [('the', 287032), ('a', 155096), ('and', 152664), ('of', 142972), ('to', 132568)]
Cinco palavras menos frequentes: [('Crocker', 1), ('McKenzieBarry', 1), ('shearer', 1), ('grossest', 1), ('unemployed...', 1)]

```

Agora

```

Cinco palavras mais frequentes: [('the', 334730), ('and', 162252), ('a', 161958), ('of', 145326), ('to', 135046)]
Cinco palavras menos frequentes: [('mckenziebarry', 1), ('grossest', 1), ('dunny', 1), ('fitzgibbon', 1), ('snacka', 1)]

```

```

1 list_treino = list(IMDB(split='train'))

1 list_word_treino_preprocessed = []
2 for target, texto in list_treino:
3     # print(texto)
4     list_word_treino_preprocessed.extend(generate_list_of_tokens(texto))
5

1 count_unknown_tokens(vocab, list_word_treino_preprocessed)

Número de tokens desconhecidas: 100714
Número total de ocorrências de tokens desconhecidas: 213699

```

Segue tabela resumo como os números:

	Tokens dif	Ocorrencias	Num médio tokens
Original	260617	566141	233.78
Preprocessada	100714	213699	232.83

Justificativa: Com o novo tokenizador que, entre outras coisas, desconsidera diferenças de caso (upper/lower) e acentos, o número de tokens distintas acaba sendo menor em uma sentença. Por conseguinte, as palavras mais frequentes do vocabulário possuem mais ocorrências nos dados de treino. E, seguindo esse raciocínio, aumentam-se as chances de uma token de um texto estar no vocabulário. Por isso, a redução no número de tokens diferentes fora do vocabulário. O tamanho médio ficou bem próximo até porque considera o número de ocorrências com repetições, sendo que o novo tokenizador se diferencia em reduzir o número de palavras distintas.

✓ **II.4.c)** Execute agora no notebook inteiro com o novo tokenizador e veja o novo valor da acurácia obtido com a melhoria do tokenizador.

Obs.: usando lr = 0.01

```

1 print(f'batch_size {batch_size}')

batch_size 128

1 train_loader = DataLoader(train_data,
2                             batch_size= batch_size,
3                             shuffle=True,
4                             num_workers=0,
5                             pin_memory=True)
6 test_loader = DataLoader(test_data,
7                             batch_size= batch_size,
8                             shuffle=False,
9                             num_workers=0,
10                            pin_memory=True)
11 print(f"train_loader tem {len(train_loader)} batches")
12 print(f"test_loader tem {len(test_loader)} batches")

train_loader tem 196 batches
test_loader tem 196 batches

```

```

1 # Model instantiation
2 model = OneHotMLP(vocab_size)
3 model = model.to(device)
4 criterion = nn.BCEWithLogitsLoss()
5 optimizer = optim.SGD(model.parameters(), lr=0.01)
6 train_start_time = time.time() # Start time of the epoch
7 test_accuracy = train_model(model, criterion, optimizer, num_epochs, train_loader, test_loader)
8
9 train_duration = time.time() - train_start_time # Duration of epoch
10
11 print(f'Treino: lr={lr}; test_accuracy={round(test_accuracy,4)} (train_duration {train_duration:.2f} sec)')

Epoch [1/5],          Loss: 0.6799,          Elapsed Time: 1.56 sec
Epoch [2/5],          Loss: 0.6358,          Elapsed Time: 1.52 sec
Epoch [3/5],          Loss: 0.6176,          Elapsed Time: 1.52 sec
Epoch [4/5],          Loss: 0.5465,          Elapsed Time: 1.56 sec
Epoch [5/5],          Loss: 0.5116,          Elapsed Time: 1.53 sec
Treino: lr=0.1; test_accuracy=81.392 (train_duration 8.98 sec)

```

✓ III - DataLoader

Vamos estudar agora o Data Loader da seção III do notebook. Em primeiro lugar anote a acurácia do notebook com as melhorias de eficiência de rodar em GPU, com ajustes de LR e do tokenizador. Em seguida mude o parâmetro shuffle na construção do objeto train_loader para False e execute novamente o notebook por completo e meça novamente a acurácia:

Tabela: Shuffle Acurácia True False

Estude o método de minimização da Loss pelo gradiente descendente utilizado em redes neurais, utilizando processamento por batches. Esse é um conceito muito importante. Veja no chatGPT qual é a relação da função Loss a ser minimizada no treinamento em função do batch size.

Exercícios:

```

1 train_loader = DataLoader(train_data,
2                           batch_size= batch_size,
3                           shuffle=False,
4                           num_workers=0,
5                           pin_memory=True)
6 test_loader = DataLoader(test_data,
7                           batch_size= batch_size,
8                           shuffle=False,
9                           num_workers=0,
10                          pin_memory=True)
11 print(f"train_loader tem {len(train_loader)} batches")
12 print(f"test_loader tem {len(test_loader)} batches")

train_loader tem 196 batches
test_loader tem 196 batches

1 # Model instantiation
2 model = OneHotMLP(vocab_size)
3 model = model.to(device)
4 criterion = nn.BCEWithLogitsLoss()
5 optimizer = optim.SGD(model.parameters(), lr=0.01)
6 train_start_time = time.time() # Start time of the epoch
7 test_accuracy = train_model(model, criterion, optimizer, num_epochs, train_loader, test_loader)
8
9 train_duration = time.time() - train_start_time # Duration of epoch
10
11 print(f'Treino: lr={lr}; test_accuracy={round(test_accuracy,4)} (train_duration {train_duration:.2f} sec)')

Epoch [1/5],          Loss: 0.2249,          Elapsed Time: 1.51 sec
Epoch [2/5],          Loss: 0.1498,          Elapsed Time: 1.51 sec
Epoch [3/5],          Loss: 0.1079,          Elapsed Time: 1.50 sec
Epoch [4/5],          Loss: 0.0859,          Elapsed Time: 1.51 sec
Epoch [5/5],          Loss: 0.0725,          Elapsed Time: 1.50 sec
Treino: lr=0.1; test_accuracy=50.0 (train_duration 8.88 sec)

```

R.: Eis a tabela:

Shuffle	Acurácia (teste)
True	81.4
False	50

✓ III.1 Batch 1

- ✓ **III.1.a)** Explique as duas principais vantagens do uso de batch no treinamento de redes neurais.

R.:

- **Eficiência Computacional:** O treinamento em batches permite processar várias amostras de dados ao mesmo tempo. Isso é mais eficiente do que atualizar os pesos do modelo após cada exemplo individual. Reduz a sobrecarga computacional, especialmente em GPUs, acelerando o treinamento.
- **Estabilidade e Convergência do Treinamento:** O cálculo do gradiente (derivada) da função de custo em relação aos parâmetros do modelo é mais estável com batches. O gradiente é uma média das amostras no lote, o que reduz a variância. Isso ajuda a evitar oscilações e convergência instável durante o treinamento.

- ✓ **III.1.b)** Explique por que é importante fazer o embaralhamento das amostras do batch em cada nova época.

R.:

1. **Redução de Viés de Aprendizado (generalização melhorada):** Quando as amostras são apresentadas ao modelo em uma ordem específica, ele pode aprender a depender da sequência. Por exemplo, se as primeiras amostras forem sempre de uma classe específica, o modelo pode se tornar tendencioso em relação a essa classe. Embaralhar as amostras garante que o modelo não seja influenciado pela ordem de apresentação. Embaralhar as amostras torna o treinamento mais robusto e ajuda o modelo a generalizar melhor para dados não vistos.
2. **Estabilidade do Treinamento:** O embaralhamento aleatório das amostras introduz uma variabilidade natural no treinamento. Isso ajuda a evitar que o modelo fique preso em mínimos locais ou em trajetórias de gradiente específicas. Também ajuda a explorar diferentes partes do espaço de parâmetros.

Em resumo, o embaralhamento das amostras do batch é essencial para garantir que o modelo aprenda de forma imparcial, seja estável durante o treinamento e generalize bem para novos dados.

- ✓ **III.1.c)** Se você alterar o `shuffle=False` no instanciamento do objeto `test_loader`, por que o cálculo da acurácia não se altera?

R.:

Como a acurácia é calculada sobre todas as previsões, a ordem em que as previsões são feitas não importa. Seja qual for a ordem em que as amostras são processadas, a acurácia final será a mesma.

✓ **III.2 Batch 2**

- ✓ **III.2.a)** Faça um laço no objeto `train_loader` e meça quantas iterações o Loader tem. Mostre o código para calcular essas iterações. Explique o valor encontrado.

```
1 """
2 O número de iterações é determinado pelo tamanho do conjunto de dados de treinamento e o tamanho do lote (batch size).
3 """
4
5 num_iterations = 0
6 for _ in train_loader:
7     num_iterations += 1
8
9 print(f'The train_loader has {num_iterations} iterations.')
10 print(f"Equivalente ao len(train_loader): {len(train_loader)} batches")
11
```

The train_loader has 196 iterations.
Equivalente ao len(train_loader): 196 batches

- ✓ **III.2.b)** Imprima o número de amostras do último batch do `train_loader` e justifique o valor encontrado? Ele pode ser menor que o `batch_size`?

```

1 """
2 O último lote pode ter menos amostras do que o tamanho do lote.
3 Isso ocorre quando (N) não é um múltiplo exato de (B).
4 """
5
6 # Encontre o número total de amostras no conjunto de dados
7 total_samples = len(train_loader.dataset)
8
9 # Calcule o número de lotes
10 num_batches = len(train_loader)
11
12 # Calcule o tamanho do último lote
13 samples_in_last_batch = total_samples % batch_size
14
15 # Imprima os resultados
16 print(f"Número total de amostras: {total_samples}")
17 print(f"Número de lotes: {num_batches}")
18 print(f"Amostras no último lote: {samples_in_last_batch}")
19
20
    Número total de amostras: 25000
    Número de lotes: 196
    Amostras no último lote: 40

1 """
2 Forma alternativa
3 """
4 last_batch_size = 0
5 for batch in train_loader:
6     last_batch_size = len(batch[0]) # Assuming batch is a tuple of (inputs, labels)
7
8 print(f'The last batch has {last_batch_size} samples.')

    The last batch has 40 samples.

```

- ✓ **III.2.c)** Calcule R, a relação do número de amostras positivas sobre o número de amostras no batch e no final encontre o valor médio de R, para ver se o data loader está entregando batches balanceados. Desta vez, em vez de fazer um laço explícito, utilize list comprehension para criar uma lista contendo a relação R de cada amostra no batch. No final, calcule a média dos elementos da lista para fornecer a resposta final.

Voltando shuffle=True

```

1 train_loader = DataLoader(train_data,
2                           batch_size= batch_size,
3                           shuffle=True,
4                           num_workers=0,
5                           pin_memory=True)
6 print(f"train_loader tem {len(train_loader)} batches")
7
    train_loader tem 196 batches

1
2 # Calculate the ratio R for each batch using a list comprehension
3 ratios = [torch.sum(labels == 1).item() / len(labels) for _, labels in train_loader]
4
5 # Calculate the average ratio
6 average_ratio = sum(ratios) / len(ratios)
7
8 print(f'Valor de R (average ratio of positive samples) is {average_ratio:.4f}')
9
    Valor de R (average ratio of positive samples) is 0.4998

```

O que demonstra que está equilibrado o número de classes positivas, graças à aleatoriedade e à proporção em todo o dataset ser de 50%.

- ✓ **III.2.d)** Mostre a estrutura de um dos batches. Cada batch foi criado no método **getitem** do Dataset, linha 20. É formado por uma tupla com o primeiro elemento sendo a codificação one-hot do texto e o segundo elemento o label esperado, indicando positivo ou negativo. Mostre o shape (linhas e colunas) e o tipo de dado (float ou integer), tanto da entrada da rede como do label esperado. Desta vez selecione um elemento do batch do train_loader utilizando as funções next e iter: batch = next(iter(train_loader)).

```

1
2 # Get the first batch from the train_loader
3 batch = next(iter(train_loader))
4
5 # The batch is a tuple of (inputs, labels)
6 inputs, labels = batch
7
8 # Print the shape and data type of the inputs and labels
9 print(f'Inputs shape: {inputs.shape}, type: {inputs.dtype}')
10 print(f'Labels shape: {labels.shape}, type: {labels.dtype}')

Inputs shape: torch.Size([128, 20001]), type: torch.float32
Labels shape: torch.Size([128]), type: torch.int64

```

✓ III.3 Batch x acurácia

Alteradas funções para tratar `batch_size == 1` `outputs.view(-1)` e `labels.view(-1)` irão remodelar `outputs` e `labels` para terem uma forma de `(n)`, onde `n` é o número total de elementos em cada tensor. Isso garantirá que `outputs` e `labels` tenham a mesma forma, independentemente do tamanho do batch.

Obs.: Quando o tamanho do batch é 1 porque a função `squeeze()` estava sendo usada para remover as dimensões de tamanho 1 do tensor de saída do seu modelo (`outputs`). Quando o tamanho do batch é 1, `outputs` tem uma dimensão de tamanho 1, então `squeeze()` remove essa dimensão, resultando em um tensor de tamanho zero. No entanto, o tensor de rótulos (`labels`) ainda tem uma dimensão de tamanho 1, então a função de perda `criterion` gera um erro porque espera que `outputs` e `labels` tenham as mesmas dimensões.

```

1 def evaluate(model, test_loader):
2     ## evaluation
3     model.eval()
4
5     with torch.no_grad():
6         correct = 0
7         total = 0
8         for inputs, labels in test_loader:
9             inputs = inputs.to(device)
10            labels = labels.to(device)
11            outputs = model(inputs)
12
13            predicted = torch.round(torch.sigmoid(outputs)).view(-1)
14            total += labels.size(0)
15            correct += (predicted == labels.view(-1)).sum().item()
16            test_accuracy = 100 * correct / total
17
18            return test_accuracy

```

```

1 def train_model(model, criterion, optimizer, num_epochs, train_loader, test_loader, verbose:bool=True):
2
3     # Training loop
4     for epoch in range(num_epochs):
5         epoch_start_time = time.time() # Start time of the epoch
6         model.train()
7         for cnt_batch, (inputs, labels) in enumerate(train_loader):
8
9
10            # print(f"Batch {cnt+1}")
11            # mover para gpu
12            inputs = inputs.to(device)
13            labels = labels.to(device)
14            # Forward pass
15            outputs = model(inputs)
16            if verbose and cnt_batch==0:
17                # Verifique as dimensões dos tensores
18                print("Dimensões dos outputs:", outputs.shape)
19                print("Dimensões dos labels:", labels.shape)
20            loss = criterion(outputs.view(-1), labels.view(-1).float())
21            # Backward and optimize
22            optimizer.zero_grad()
23            loss.backward()
24            optimizer.step()
25
26            epoch_duration = time.time() - epoch_start_time # Duration of epoch
27
28            if verbose:
29                print(f'Epoch [{epoch+1}/{num_epochs}], \
30                    Loss: {loss.item():.4f}, \
31                    Elapsed Time: {epoch_duration:.2f} sec')
32
33            # Evaluate the model and record the test accuracy
34            test_accuracy = evaluate(model, test_loader)
35
36            return test_accuracy
37
38

```

- ✓ **III.3.a)** Verifique a influência do batch size na acurácia final do modelo. Experimente usar um batch size de 1 amostra apenas e outro com mais de 128 e comente sobre os resultados.

```

1 num_values = 8
2 # Define the minimum and maximum learning rate and the number of values
3 min_batch_size = 1
4 max_batch_size = 512
5 batch_size_list = np.linspace(min_batch_size, max_batch_size, num=num_values)
6 # Converta os valores para inteiros
7 batch_size_list = [int(size) for size in batch_size_list]
8 print('linspace', batch_size_list)
9
10

```

linspace [1, 74, 147, 220, 293, 366, 439, 512]

Gerada uma lista de todas as potências de 2 até max_batch_size, incluindo 1, você pode usar a função pow do Python.

```

1
2 # Define the maximum batch size
3 max_batch_size = 1024
4
5 # Generate a list of all powers of 2 up to max_batch_size, including 1
6 batch_size_list = [pow(2, i) for i in range(int(np.log2(max_batch_size)) + 1)]
7
8 print('Powers of 2 up to max_batch_size:', batch_size_list)

```

Powers of 2 up to max_batch_size: [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]

```

1 test_loader = DataLoader(test_data,
2                           batch_size= 128,
3                           shuffle=False,
4                           num_workers=0,
5                           pin_memory=True)
6 print(f"test_loader tem {len(test_loader)} batches")
7

```

test_loader tem 196 batches

```

1 # Initialize a dictionary to record test accuracy for each learning rate
2 test_accuracies_by_batch_size = {}
3 train_duration_by_batch_size = {}
4
5 for cnt_treino, batch_size in enumerate(batch_size_list):
6     train_loader = DataLoader(train_data,
7                               batch_size=batch_size,
8                               shuffle=True,
9                               num_workers=0,
10                              pin_memory=True)
11     # Model instantiation
12     model = OneHotMLP(vocab_size)
13     model = model.to(device)
14     criterion = nn.BCEWithLogitsLoss()
15     optimizer = optim.SGD(model.parameters(), lr=0.01)
16     train_start_time = time.time() # Start time of the epoch
17     test_accuracy = train_model(model, criterion, optimizer, 5, train_loader, test_loader, verbose=False)
18
19     train_duration = time.time() - train_start_time # Duration of epoch
20
21     test_accuracies_by_batch_size[batch_size] = test_accuracy
22     train_duration_by_batch_size[batch_size] = train_duration
23
24     print(f'Treino [{cnt_treino + 1}/{len(batch_size_list)}]: batch_size={batch_size}; Número de batches de treino: {len(train_loader)}')
25
26 print(f'test_accuracies_by_batch_size {test_accuracies_by_batch_size}')

```

```

Treino [1/11]: batch_size=1; Número de batches de treino: 25000; test_accuracy=87.076 (train_duration 106.19 sec)
Treino [2/11]: batch_size=2; Número de batches de treino: 12500; test_accuracy=86.828 (train_duration 58.53 sec)
Treino [3/11]: batch_size=4; Número de batches de treino: 6250; test_accuracy=86.74 (train_duration 34.31 sec)
Treino [4/11]: batch_size=8; Número de batches de treino: 3125; test_accuracy=87.98 (train_duration 22.60 sec)
Treino [5/11]: batch_size=16; Número de batches de treino: 1563; test_accuracy=88.076 (train_duration 14.99 sec)
Treino [6/11]: batch_size=32; Número de batches de treino: 782; test_accuracy=87.492 (train_duration 11.85 sec)
Treino [7/11]: batch_size=64; Número de batches de treino: 391; test_accuracy=85.224 (train_duration 9.95 sec)
Treino [8/11]: batch_size=128; Número de batches de treino: 196; test_accuracy=81.68 (train_duration 9.01 sec)
Treino [9/11]: batch_size=256; Número de batches de treino: 98; test_accuracy=79.28 (train_duration 8.37 sec)
Treino [10/11]: batch_size=512; Número de batches de treino: 49; test_accuracy=75.968 (train_duration 7.92 sec)
Treino [11/11]: batch_size=1024; Número de batches de treino: 25; test_accuracy=71.864 (train_duration 7.90 sec)
test_accuracies_by_batch_size {1: 87.076, 2: 86.828, 4: 86.74, 8: 87.98, 16: 88.076, 32: 87.492, 64: 85.224, 128: 81.68, 256: 79.28,

```

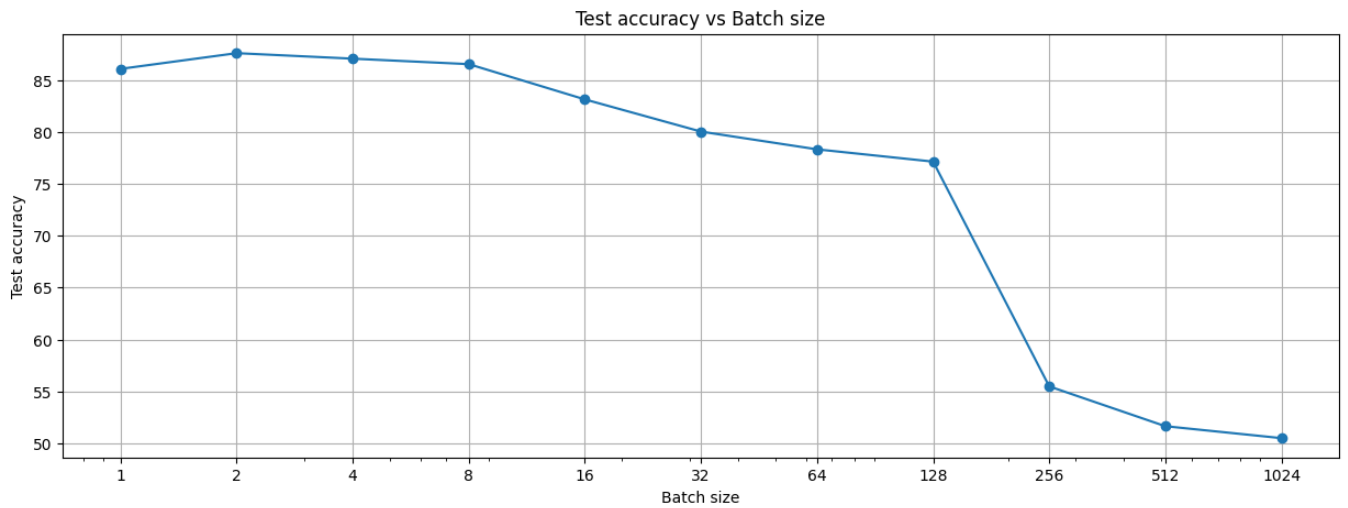


Gerando gráficos

```

1 # Plot test accuracy for each learning rate
2 plt.figure(figsize=(15, 5))
3 plt.plot(list(test_accuracies_by_batch_size.keys()), list(test_accuracies_by_batch_size.values()), marker='o')
4 plt.xscale('log')
5 plt.xlabel('Batch size')
6 plt.ylabel('Test accuracy')
7 plt.title('Test accuracy vs Batch size')
8 plt.grid(True)
9
10 # Set xticks to all batch sizes
11 plt.xticks(list(test_accuracies_by_batch_size.keys()), list(test_accuracies_by_batch_size.keys()))
12
13 plt.show()

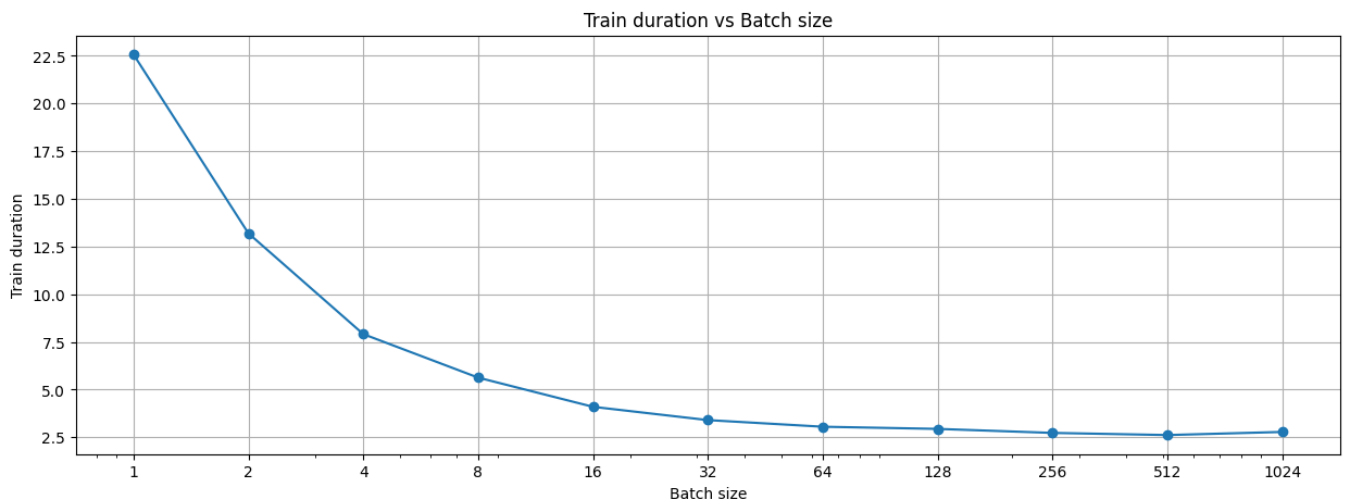
```



```

1 # Plot test accuracy for each learning rate
2 plt.figure(figsize=(15, 5))
3 plt.plot(list(train_duration_by_batch_size.keys()), list(train_duration_by_batch_size.values()), marker='o')
4 plt.xscale('log')
5 plt.xlabel('Batch size')
6 plt.ylabel('Train duration')
7 plt.title('Train duration vs Batch size')
8 plt.grid(True)
9
10 # Set xticks to all batch sizes
11 plt.xticks(list(train_duration_by_batch_size.keys()), list(train_duration_by_batch_size.keys()))
12
13 plt.show()

```



A acurácia ficou maior para batchs menores. Isso pode ser explicado:

- Gradientes mais precisos:** Quando o tamanho do batch é menor, o gradiente calculado em cada etapa do treinamento é uma estimativa menos precisa do gradiente verdadeiro. Isso pode introduzir mais ruído no processo de treinamento, o que pode, paradoxalmente, ajudar o modelo a evitar mínimos locais e encontrar melhores soluções. Em contraste, quando o tamanho do batch é maior, o gradiente é uma estimativa mais precisa, mas isso pode fazer com que o modelo fique preso em mínimos locais.
- Mais atualizações de modelo:** Quando o tamanho do batch é menor, o modelo é atualizado com mais frequência. Por exemplo, se você tem 1000 exemplos de treinamento, um tamanho de batch de 1 resultará em 1000 atualizações de modelo por época, enquanto um tamanho de batch de 100 resultará em apenas 10 atualizações de modelo por época. Mais atualizações de modelo podem permitir que o modelo aprenda mais a partir dos dados.

- 3. **Regularização implícita:** O uso de tamanhos de batch menores também pode ter um efeito de regularização, ajudando a prevenir o overfitting. Isso ocorre porque o ruído introduzido pela estimativa do gradiente com menos exemplos pode ajudar a evitar que o modelo se ajuste demais aos dados de treinamento.

No entanto, vale a pena notar que embora tamanhos de batch menores possam às vezes resultar em uma acurácia de teste maior, eles também podem tornar o treinamento mais lento (gráfico 2), porque menos exemplos são processados simultaneamente. Além disso, tamanhos de batch muito pequenos podem resultar em estimativas de gradiente muito ruidosas, o que pode tornar o treinamento instável. Portanto, a escolha do tamanho do batch é um compromisso e pode requerer alguma experimentação para encontrar o melhor valor para um determinado problema.

✓ IV - Modelo MLP

A célula da seção IV - Modelo é provavelmente uma das mais difíceis de entender, juntamente com a seção V - Treinamento, pois são onde aparecem as principais funções do PyTorch.

Iremos utilizar uma rede neural de duas camadas ditas MLP (Multi-Layer Perceptron). São duas camadas lineares, fc1 e fc2. Essas camadas também são denominadas fully connected para diferenciar de camadas convolucionais. As camadas são onde estão os parâmetros (weights) da rede neural. É importante estudar como estas camadas lineares funcionam, elas são compostas de neurônios que fazem uma média ponderada pelos parâmetros W_i mais uma constante B_i . Esses parâmetros são treinados para minimizar a função de Loss. Uma função não linear é colocada entre as camadas lineares. No caso, usamos a função ReLU (Rectified Linear Unit).

Para entender o código da célula do Modelo MLP é fundamental conhecer os conceitos de orientação a objetos do Python. O modelo é definido pela classe OneHotMLP e é instanciado no objeto model na linha 16 que implementa o modelo da rede neural, recebendo uma entrada no formato one-hot e retornando o logito para ser posteriormente convertido em probabilidade do frase ser positiva ou negativa. O método forward será chamado automaticamente quando o objeto model for usado como função. Esses modelos são projetados para processar um batch de entrada de cada vez no formato devolvido pelo Data Loader visto na seção III (Exercício III.2.d)

✓ IV.1 Exercícios para experimentar o modelo

- IV.1.a)** Faça a predição do modelo utilizando um batch do train_loader: extraia um batch do train_loader, chame de (input, target), onde input é a entrada da rede e target é o label esperado. Como a rede está com seus parâmetros (weights) aleatórios, o logito de saída da rede será um valor aleatório, porém a chamada irá executar sem erros:

```
logit = model( input)
```

aplique a função sigmoidal ao logito para convertê-lo numa probabilidade de valor entre 0 e 1.

```
1 train_loader = DataLoader(train_data,
2                             batch_size= 8,
3                             shuffle=True,
4                             num_workers=0,
5                             pin_memory=True)
6 # Model instantiation
7 model = OneHotMLP(vocab_size)
8 model = model.to(device)
9
```

```

1
2 # Obtenha o primeiro batch do train_loader
3 inputs, targets = next(iter(train_loader))
4
5 # Mova os inputs e targets para o dispositivo onde o modelo está
6 inputs = inputs.to(device)
7 targets = targets.to(device).view(-1)
8
9 # Passe os inputs pelo modelo
10 # Isso retorna os logits, que são os valores brutos de saída do modelo.
11 # Como o modelo ainda não foi treinado, esses valores são inicialmente aleatórios.
12 logits = model(inputs)
13
14 # Aplique a função sigmoid aos logits
15 # A função sigmoid mapeia qualquer número real para o intervalo (0, 1),
16 # então as probabilidades resultantes serão valores entre 0 e 1.
17 # Essas probabilidades representam a saída do modelo: a probabilidade prevista
18 # de que cada exemplo de entrada pertença à classe positiva.
19 probabilities = torch.sigmoid(logits).view(-1)
20
21 predicted = torch.round(probabilities).view(-1)
22 total = targets.size(0)
23
24 correct = (predicted == targets)
25 total_correct = correct.sum().item()
26 accuracy = 100 * total_correct / total
27
28 print(f'logits {logits}')
29 print(f'probabilities {probabilities}')
30 print(f'predicted {predicted}')
31 print(f'targets {targets}')
32 print(f'correct {correct}')
33 print(f'correct {total_correct} of {total}')
34 print(f'accuracy {accuracy}')
35
logits tensor([[0.0377],
               [0.0645],
               [0.0222],
               [0.0132],
               [0.0496],
               [0.0317],
               [0.0464],
               [0.0325]], device='cuda:0', grad_fn=<AddmmBackward0>)
probabilities tensor([0.5094, 0.5161, 0.5056, 0.5033, 0.5124, 0.5079, 0.5116, 0.5081],
                    device='cuda:0', grad_fn=<ViewBackward0>)
predicted tensor([1., 1., 1., 1., 1., 1., 1., 1.], device='cuda:0',
                grad_fn=<ViewBackward0>)
targets tensor([0, 1, 1, 0, 1, 1, 0, 0], device='cuda:0')
correct tensor([False,  True,  True, False,  True,  True, False, False],
               device='cuda:0')
correct 4 of 8
accuracy 50.0

```

IV.1.b) Agora, treine a rede executando o notebook todo e verifique se a acurácia está alta. Agora repita o exercício anterior, porém agora, compare o valor da probabilidade encontrada com o target esperado e verifique se ele acertou. Você pode considerar que se a probabilidade for maior que 0.5, pode-se dar o label 1 e se for menor que 0.5, o label 0. Observe isso que é feito na linha 11 da seção VI - Avaliação.

Se você der um print no modelo: `print(model)`, você obterá: `OneHotMLP((fc1): Linear(in_features=20001, out_features=200, bias=True) (fc2): Linear(in_features=200, out_features=1, bias=True) (relu): ReLU())`

Os pesos da primeira camada podem ser visualizados com: `model.fc1.weight` e o elemento constante (bias) pode ser visualizado com: `model.fc1.bias`. Calcule o número de parâmetros do modelo, preenchendo a seguinte tabela (utilize `shape` para verificar a estrutura de cada parâmetro do modelo):

Tabela: layer fc1 fc2 TOTAL weight bias weight bias size

Treino completo

```

1 test_loader = DataLoader(test_data,
2                           batch_size= 8,
3                           shuffle=False,
4                           num_workers=0,
5                           pin_memory=True)
6 print(f"test_loader tem {len(test_loader)} batches")
7

```


test_loader tem 3125 batches

```
1 # Initialize a dictionary to record test accuracy for each learning rate
2
3 train_loader = DataLoader(train_data,
4                             batch_size= 8,
5                             shuffle=True,
6                             num_workers=0,
7                             pin_memory=True)
8 # Model instantiation
9 model = OneHotMLP(vocab_size)
10 model = model.to(device)
11 criterion = nn.BCEWithLogitsLoss()
12 optimizer = optim.SGD(model.parameters(), lr=0.01)
13 train_start_time = time.time() # Start time of the epoch
14 test_accuracy = train_model(model, criterion, optimizer, 5, train_loader, test_loader, verbose=False)
15
16 train_duration = time.time() - train_start_time # Duration of epoch
17
18
19 print(f'Treino [{cnt_treino + 1}/{len(batch_size_list)}]: batch_size={batch_size}; Número de batches de treino: {len(train_loader)};
20
21
```

Treino [11/11]: batch_size=1024; Número de batches de treino: 3125; test_accuracy=87.732 (train_duration 22.75 sec)

Verificação da acurácia após treino

```
1 list_data_loader = [{'treino':train_loader}, {'teste':test_loader, }]
2 # Obtenha o primeiro batch do train_loader
3 for data_loader_avaliacao in list_data_loader:
4     #print(data_loader_avaliacao)
5     nome_data_loader = list(data_loader_avaliacao.keys())[0]
6     #print(nome_data_loader)
7     data_loader = data_loader_avaliacao[nome_data_loader]
8     #print(data_loader)
9     inputs, targets = next(iter(data_loader))
10
11 # Mova os inputs e targets para o dispositivo onde o modelo está
12 inputs = inputs.to(device)
13 targets = targets.to(device).view(-1)
14
15 # Passe os inputs pelo modelo
16 # Isso retorna os logits, que são os valores brutos de saída do modelo.
17 # Como o modelo ainda não foi treinado, esses valores são inicialmente aleatórios.
18 logits = model(inputs)
19
20 # Aplique a função sigmoid aos logits
21 # A função sigmoid mapeia qualquer número real para o intervalo (0, 1),
22 # então as probabilidades resultantes serão valores entre 0 e 1.
23 # Essas probabilidades representam a saída do modelo: a probabilidade prevista
24 # de que cada exemplo de entrada pertença à classe positiva.
25 probabilities = torch.sigmoid(logits).view(-1)
26
27 predicted = torch.round(probabilities).view(-1)
28 total = targets.size(0)
29
30 correct = (predicted == targets)
31 total_correct = correct.sum().item()
32 accuracy = 100 * total_correct / total
33
34 print(f'\n\nEm ambiente de: {nome_data_loader}')
35
36 print(f'logits {logits}')
37 print(f'probabilities {probabilities}')
38 print(f'predicted {predicted}')
39 print(f'targets {targets}')
40 print(f'correct {correct}')
41 print(f'correct {total_correct} of {total}')
42 print(f'accuracy {accuracy}')
43
```

Em ambiente de: treino
logits tensor([[-7.3362],
[0.6910],
[4.8479],
[-4.7868],
[0.9367],
[7.0848],

```

        [ 0.8461],
        [ 4.5086]], device='cuda:0', grad_fn=<AddmmBackward0>)
probabilities tensor([6.5110e-04, 6.6619e-01, 9.9222e-01, 8.2704e-03, 7.1844e-01, 9.9916e-01,
        6.9975e-01, 9.8911e-01], device='cuda:0', grad_fn=<ViewBackward0>)
predicted tensor([0., 1., 1., 0., 1., 1., 1., 1.], device='cuda:0',
        grad_fn=<ViewBackward0>)
targets tensor([0, 1, 1, 0, 1, 1, 1, 1], device='cuda:0')
correct tensor([True, True, True, True, True, True, True, True], device='cuda:0')
correct 8 of 8
accuracy 100.0

```

```

Em ambiente de: teste
logits tensor([[ 7.8473],
        [ 1.2208],
        [ 0.1196],
        [ 9.2927],
        [-5.5199],
        [ 3.4430],
        [ 2.6541],
        [ 7.4055]], device='cuda:0', grad_fn=<AddmmBackward0>)
probabilities tensor([0.9996, 0.7722, 0.5299, 0.9999, 0.0040, 0.9690, 0.9343, 0.9994],
        device='cuda:0', grad_fn=<ViewBackward0>)
predicted tensor([1., 1., 1., 1., 0., 1., 1., 1.], device='cuda:0',
        grad_fn=<ViewBackward0>)
targets tensor([1, 1, 1, 1, 1, 1, 1, 1], device='cuda:0')
correct tensor([ True,  True,  True,  True, False,  True,  True,  True],
        device='cuda:0')
correct 7 of 8
accuracy 87.5

```

Observação Como esperado, a acurácia nos dados de treinamento alcançou 100%, mas em teste 87.5%. O modelo ficou especialista nos dados de treinamento.

Obtendo o total dos parâmetros

```

1 print(model)

OneHotMLP(
  (fc1): Linear(in_features=20001, out_features=200, bias=True)
  (fc2): Linear(in_features=200, out_features=1, bias=True)
  (relu): ReLU()
)

1 model.fc1.weight.size(), model.fc1.bias.size()

(torch.Size([200, 20001]), torch.Size([200]))

1
2 # Initialize dictionaries to hold the counts and shapes
3 param_counts = {}
4 param_shapes = {}
5 layer_counts = {}
6
7 # Iterate over each parameter
8 for name, param in model.named_parameters():
9     # Update the count and shape for this parameter
10    param_counts[name] = param.numel()
11    param_shapes[name] = list(param.size())
12
13    # Get the layer name and update the layer count
14    layer_name = name.split('.')[0]
15    layer_counts[layer_name] = layer_counts.get(layer_name, 0) + param.numel()
16
17 # Print the number of parameters and shape for each weight and bias in each layer
18 for param_name in param_counts.keys():
19    print(f'{param_name}: {param_counts[param_name]} parameters, shape {param_shapes[param_name]}')
20
21 # Print the total number of parameters in each layer
22 for layer_name in layer_counts.keys():
23    print(f'{layer_name}: {layer_counts[layer_name]} total parameters')
24
25 # Calculate and print the total number of parameters in the model
26 total_params = sum(param_counts.values())
27 print(f'Total parameters in the model: {total_params}')
28

fc1.weight: 4000200 parameters, shape [200, 20001]
fc1.bias: 200 parameters, shape [200]
fc2.weight: 200 parameters, shape [1, 200]
fc2.bias: 1 parameters, shape [1]
fc1: 4000400 total parameters

```

fc2: 201 total parameters
Total parameters in the model: 4000601

layer					
	weight	total	bias	total	total
fc1	[200, 20001]	4000200	[200]	200	4000400
fc2	[1, 200]	200	[1]	1	201
total		4000400		201	4000601

✓ V - Treinamento

Agora vamos entrar na principal seção do notebook que minimiza a Loss para ajustar os pesos do modelo.

Cálculo da Loss

A Loss é uma comparação entre a saída do modelo e o label (target). A Loss mais utilizada para problemas de classificação é a Entropia Cruzada. A equação da entropia cruzada para o caso binário (2 classes: 0 ou 1; True ou False) é dada por:

$$H = -\frac{1}{N} \sum_{i=1}^N y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

Nessa fórmula:

(H) é a entropia cruzada. (N) é o número de amostras. (y_i) é o rótulo verdadeiro da amostra (i). (p_i) é a probabilidade prevista pelo modelo para a amostra (i).

Muitas vezes chamamos y_i de target e p_i (y_i)^{de prob.}

Quando a Loss é zero, significa que o modelo está predizendo tanto as amostras positivas como as amostras negativas com probabilidade de 100%. O objetivo é otimizar o modelo para conseguir minimizar a Loss ao máximo.

A rede neural é o nosso modelo que recebe a entrada com um batch de amostras e retorna um batch de logits ou output.

`output = model(input)`

para converter o logito (output) em probabilidade, utiliza-se a função sigmóide que é dada pela equação específica

Assim, o código pytorch para estimarmos a probabilidade de um texto codificado no formato one-hot na variável input pode ser:

`prob = torch.sigmoid(output)`

Atenção: observe que esses comandos estão processando todas as amostras no batch, que nesse notebook tem 128 amostras no batch size.

✓ V.1 Exercícios:

V.1.a) Qual é o valor teórico da Loss quando o modelo não está treinado, mas apenas inicializado? Isto é, a probabilidade

- ✓ predita tanto para a classe 0 como para a classe 1, é sempre 0,5? Justifique. Atenção: na equação da Entropia Cruzada utilize o logaritmo natural.

R.: A função de perda de entropia cruzada binária, que é comumente usada para problemas de classificação binária, é definida como:

$$\text{BCELoss} = -[y * \log(p) + (1 - y) * \log(1 - p)]$$

onde y é o rótulo verdadeiro (0 ou 1) e p é a probabilidade prevista para a classe 1.

Quando o modelo é apenas inicializado e não treinado, e se assumirmos que ele está prevendo uma probabilidade de 0,5 para ambas as classes (o que seria o caso se o modelo estivesse prevendo aleatoriamente), então a função de perda de entropia cruzada binária se simplifica para:

$$\begin{aligned} \text{BCELoss} &= -[y * \log(0.5) + (1 - y) * \log(0.5)] \\ &= -\log(0.5) \\ &= \log(2) \end{aligned}$$

O logaritmo natural de 2 é aproximadamente 0.6931. Portanto, o valor teórico da perda quando o modelo não está treinado e está prevendo uma probabilidade de 0,5 para ambas as classes é aproximadamente 0.6931.

Obs.:note que isso é apenas uma aproximação teórica. Na prática, um modelo recém-inicializado pode não prever exatamente 0,5 para todas as entradas, dependendo de como seus pesos são inicializados. Além disso, a perda real também dependerá da distribuição dos rótulos verdadeiros y no conjunto de dados.

- ✓ **V.1.b)** Utilize as amostras do primeiro batch: `(input,target) = next(iter(train_loader))` e calcule o valor da Loss utilizando a equação fornecida anteriormente utilizando o pytorch. Verifique se este valor confere com o valor teórico do exercício anterior.

```
1 # Suponha que você já tenha definido train_data, vocab_size e OneHotMLP
2
3 train_loader = DataLoader(train_data,
4                             batch_size=128,
5                             shuffle=True,
6                             num_workers=0,
7                             pin_memory=True)
8
9 # Instanciação do modelo
10 model = OneHotMLP(vocab_size)
11 model = model.to(device)
12
13 # Obtenha o primeiro batch do train_loader
14 inputs, targets = next(iter(train_loader))
15
16 # Mova os inputs e targets para o dispositivo onde o modelo está
17 inputs = inputs.to(device)
18 targets = targets.to(device).view(-1)
19
20 # Passe os inputs pelo modelo
21 # Isso retorna os logits, que são os valores brutos de saída do modelo.
22 # Como o modelo ainda não foi treinado, esses valores são inicialmente aleatórios.
23 logits = model(inputs)
24
25 # Aplique a função sigmoid aos logits
26 # A função sigmoid mapeia qualquer número real para o intervalo (0, 1),
27 # então as probabilidades resultantes serão valores entre 0 e 1.
28 # Essas probabilidades representam a saída do modelo: a probabilidade prevista
29 # de que cada exemplo de entrada pertença à classe positiva.
30 probabilities = torch.sigmoid(logits).view(-1)
31
32 # Calcule a entropia cruzada manualmente
33 # A fórmula é: - (y * log(p) + (1 - y) * log(1 - p))
34 # onde y é a classe real (0 ou 1) e p é a probabilidade prevista
35 loss = -(targets * torch.log(probabilities) + (1 - targets) * torch.log(1 - probabilities)).mean()
36
37 print(f'loss {loss.item()}')
38
39
40 loss 0.6965450048446655
```

R: valor confere com o teórico

- ✓ **V.1.c)** O pytorch possui várias funções que facilitam o cálculo da Loss pela Entropia Cruzada. Utilize a classe `nn.BCELoss` (Binary Cross Entropy Loss). Você primeiro deve instanciar uma função da classe `nn.BCELoss`. Esta função instanciada recebe dois parâmetros (`probs`, `targets`) e retorna a Loss. Use a busca do Google para ver a documentação do `BCELoss` do pytorch.

Calcule então a função de Loss da entropia cruzada, porém usando agora a função instanciada pelo `BCELoss` e confira se o resultado é exatamente o mesmo obtido no exercício anterior.

Para usar a classe `nn.BCELoss` (Binary Cross Entropy Loss) em vez de `nn.BCEWithLogitsLoss`, você precisa aplicar a função sigmoid aos logits antes de passá-los para a função de perda.

Neste exemplo, a linha `loss = criterion(probabilities, targets.float())` aplica a função de perda `nn.BCELoss` às probabilidades, não aos logits. A função `nn.BCELoss` espera que as entradas sejam probabilidades entre 0 e 1, então você precisa aplicar a função sigmoid aos logits antes de passá-los para a função de perda.

```

1 train_loader = DataLoader(train_data,
2                             batch_size= 128,
3                             shuffle=True,
4                             num_workers=0,
5                             pin_memory=True)
6 # Model instantiation
7 model = OneHotMLP(vocab_size)
8 model = model.to(device)
9 criterion = nn.BCELoss()
10
11 # Obtenha o primeiro batch do train_loader
12 inputs, targets = next(iter(train_loader))
13
14 # Mova os inputs e targets para o dispositivo onde o modelo está
15 inputs = inputs.to(device)
16 targets = targets.to(device).view(-1)
17
18 # Passe os inputs pelo modelo
19 # Isso retorna os logits, que são os valores brutos de saída do modelo.
20 # Como o modelo ainda não foi treinado, esses valores são inicialmente aleatórios.
21 logits = model(inputs)
22
23 # Aplique a função sigmoid aos logits
24 # A função sigmoid mapeia qualquer número real para o intervalo (0, 1),
25 # então as probabilidades resultantes serão valores entre 0 e 1.
26 # Essas probabilidades representam a saída do modelo: a probabilidade prevista
27 # de que cada exemplo de entrada pertença à classe positiva.
28 probabilities = torch.sigmoid(logits).view(-1)
29
30 predicted = torch.round(probabilities).view(-1)
31 total = targets.size(0)
32
33 correct = (predicted == targets)
34 total_correct = correct.sum().item()
35 accuracy = 100 * total_correct / total
36
37 # Apply the loss function to the probabilities, not the logits
38 loss = criterion(probabilities, targets.float())
39
40 print(f'loss {loss.item()}')
41
42
43     loss 0.6915096044540405

```

- ✓ **V.1.d)** Repita o mesmo exercício, porém agora usando a classe `nn.BCEWithLogitsLoss`, que é a opção utilizada no notebook. O resultado da Loss deve igualar aos resultados anteriores.

Minimização da Loss pelo gradiente descendente

Estude o método do gradiente descendente para minimizar uma função. Como curiosidade, pergunte ao chatGPT quando este método de minimização foi usado pela primeira vez. Aproveite e peça para ele explicar o método de uma maneira bem simples e ilustrativa. Peça para ele explicar qual é a forma moderna de se calcular computacionalmente o gradiente de uma função. Finalmente peça para ele explicar as linhas 3, 6, e (20, 21 e 22) do laço de treinamento.

```

1 train_loader = DataLoader(train_data,
2                             batch_size= 128,
3                             shuffle=True,
4                             num_workers=0,
5                             pin_memory=True)
6 # Model instantiation
7 model = OneHotMLP(vocab_size)
8 model = model.to(device)
9 criterion = nn.BCEWithLogitsLoss()
10
11 # Obtenha o primeiro batch do train_loader
12 inputs, targets = next(iter(train_loader))
13
14 # Mova os inputs e targets para o dispositivo onde o modelo está
15 inputs = inputs.to(device)
16 targets = targets.to(device).view(-1)
17
18 # Passe os inputs pelo modelo
19 # Isso retorna os logits, que são os valores brutos de saída do modelo.
20 # Como o modelo ainda não foi treinado, esses valores são inicialmente aleatórios.
21 logits = model(inputs)
22
23 # Aplique a função sigmoid aos logits
24 # A função sigmoid mapeia qualquer número real para o intervalo (0, 1),
25 # então as probabilidades resultantes serão valores entre 0 e 1.
26 # Essas probabilidades representam a saída do modelo: a probabilidade prevista
27 # de que cada exemplo de entrada pertença à classe positiva.
28 probabilities = torch.sigmoid(logits).view(-1)
29
30 predicted = torch.round(probabilities).view(-1)
31 total = targets.size(0)
32
33 correct = (predicted == targets)
34 total_correct = correct.sum().item()
35 accuracy = 100 * total_correct / total
36
37 loss = criterion(logits.view(-1), targets.float())
38
39 print(f'loss {loss.item()}')
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

✓ V.2 Exercícios:

- ✓ **V.2.a)** Modifique a célula do laço de treinamento de modo que a primeira Loss a ser impressa seja a Loss com o modelo inicializado (isto é, sem nenhum treinamento), fornecendo a Loss esperada conforme os exercícios feitos anteriormente. Observe que desta forma, fica fácil verificar se o seu modelo está correto e a Loss está sendo calculada corretamente.

Atenção: Mantenha esse código da impressão do valor da Loss inicial, antes do treinamento, nesta célula, pois ela é sempre útil para verificar se não tem nada errado, antes de começar o treinamento.

```

1 criterion.__dict__

{
  'training': True,
  '_parameters': OrderedDict(),
  '_buffers': OrderedDict([('weight', None), ('pos_weight', None)]),
  '_non_persistent_buffers_set': set(),
  '_backward_pre_hooks': OrderedDict(),
  '_backward_hooks': OrderedDict(),
  '_is_full_backward_hook': None,
  '_forward_hooks': OrderedDict(),
  '_forward_hooks_with_kwargs': OrderedDict(),
  '_forward_hooks_always_called': OrderedDict(),
  '_forward_pre_hooks': OrderedDict(),
  '_forward_pre_hooks_with_kwargs': OrderedDict(),
  '_state_dict_hooks': OrderedDict(),
  '_state_dict_pre_hooks': OrderedDict(),
  '_load_state_dict_pre_hooks': OrderedDict(),
  '_load_state_dict_post_hooks': OrderedDict(),
  '_modules': OrderedDict(),
  'reduction': 'mean'
}

```

```

1 def validar_correcao_loss_esperada_xentropy(model, criterion, train_loader):
2
3     # Forward pass com o modelo inicializado
4     # Obtenha o primeiro batch do train_loader
5     inputs, targets = next(iter(train_loader))
6
7     # Mova os inputs e targets para o dispositivo onde o modelo está
8     inputs = inputs.to(device)
9     targets = targets.to(device).view(-1)
10
11     initial_outputs = model(inputs) # Inputs devem ser do DataLoader de treinamento
12     loss = criterion(initial_outputs.view(-1), targets.float())
13
14     # Valor esperado da Loss
15     valor_esperado = 0.6931
16
17     diff = abs(loss.item()-valor_esperado)
18     # Verifique se a Loss inicial está próxima do valor esperado
19     if diff < 0.01: # Considerando igualdade até 2 casas decimais
20         print(f"A Loss inicial de cross entropy {loss} está correta! Diferença de {diff:.4f} (arredondamentos)")
21         return True
22     else:
23         print(f"A Loss inicial de cross entropy não corresponde ao valor esperado. Valor atual: {loss.item():.4f}. Valor esperado par
24         return False
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

A Loss inicial de cross entropy 0.693384051322937 está correta! Diferença de 0.0003 (arredondamentos)

True

```

1 def train_model(model, criterion, optimizer, num_epochs, train_loader, test_loader, verbose:bool=True):
2
3     if validar_correcao_loss_esperada_xentropy(model, criterion, train_loader):
4         # Training loop
5         for epoch in range(num_epochs):
6             epoch_start_time = time.time() # Start time of the epoch
7             model.train()
8             for cnt_batch, (inputs, labels) in enumerate(train_loader):
9
10
11                 # print(f"Batch {cnt+1}")
12                 # mover para gpu
13                 inputs = inputs.to(device)
14                 labels = labels.to(device)
15                 # Forward pass
16                 outputs = model(inputs)
17                 if verbose and cnt_batch==0:
18                     # Verifique as dimensões dos tensores
19                     print("Dimensões dos outputs:", outputs.shape)
20                     print("Dimensões dos labels:", labels.shape)
21                 loss = criterion(outputs.view(-1), labels.view(-1).float())
22                 # Backward and optimize
23                 optimizer.zero_grad()
24                 loss.backward()
25                 optimizer.step()
26
27             epoch_duration = time.time() - epoch_start_time # Duration of epoch
28
29             if verbose:
30                 print(f'Epoch [{epoch+1}/{num_epochs}], \
31                     Loss: {loss.item():.4f}, \
32                     Elapsed Time: {epoch_duration:.2f} sec')
33
34             # Evaluate the model and record the test accuracy
35             test_accuracy = evaluate(model, test_loader)
36
37             return test_accuracy
38
39

```

V.2.b) Execute a célula de treinamento por uma segunda vez e observe que a Loss continua diminuindo e o modelo está

- ✓ continuando a ser treinado. O que é necessário fazer para que o treinamento comece novamente do modelo aleatório? Qual(is) célula(s) é(são) preciso executar antes de executar o laço de treinamento novamente?

```

1 train_loader = DataLoader(train_data,
2                           batch_size= 8,
3                           shuffle=True,
4                           num_workers=0,
5                           pin_memory=True)
6 # Model instantiation
7 model = OneHotMLP(vocab_size)
8 model = model.to(device)
9 criterion = nn.BCEWithLogitsLoss()
10 optimizer = optim.SGD(model.parameters(), lr=0.01)
11 train_start_time = time.time() # Start time of the epoch
12 test_accuracy = train_model(model, criterion, optimizer, 1, train_loader, test_loader, verbose=False)
13 test_accuracy = train_model(model, criterion, optimizer, 1, train_loader, test_loader, verbose=False)
14

```

A Loss inicial de cross entropy 0.6963748931884766 está correta! Diferença de 0.0033 (arredondamentos)

A Loss inicial de cross entropy não corresponde ao valor esperado. Valor atual: 0.1784. Valor esperado para loss: 0.6931. Diferença



Realmente não corresponde à inicial, pois os parâmetros do modelo foram atualizados e ela diminuiu (houve aprendizado). Para recomeçar em modo aleatório é necessário reiniciar os parâmetros, recriando-se o modelo. Conforme abaixo:


```

1 train_loader = DataLoader(train_data,
2                             batch_size= 8,
3                             shuffle=True,
4                             num_workers=0,
5                             pin_memory=True)
6 # Model instantiation
7 model = OneHotMLP(vocab_size)
8 model = model.to(device)
9 criterion = nn.BCEWithLogitsLoss()
10 optimizer = optim.SGD(model.parameters(), lr=0.01)
11 train_start_time = time.time() # Start time of the epoch
12 test_accuracy = train_model(model, criterion, optimizer, 1, train_loader, test_loader, verbose=False)
13 model = OneHotMLP(vocab_size)
14 model = model.to(device)
15 optimizer = optim.SGD(model.parameters(), lr=0.01)
16 # optimizer = optim.SGD(model.parameters(), lr=0.01)
17 test_accuracy = train_model(model, criterion, optimizer, 1, train_loader, test_loader, verbose=False)
18
    A Loss inicial de cross entropy 0.6940696239471436 está correta! Diferença de 0.0010 (arredondamentos)
    A Loss inicial de cross entropy 0.6977269053459167 está correta! Diferença de 0.0046 (arredondamentos)

```

✓ V.3 Exercícios:

Modificando a rede para gerar dois logitos no lugar de 1

Existe uma forma alternativa de implementar um modelo binário utilizando 2 logitos, um para dar a probabilidade da classe positiva e outro para a classe negativa. Para isso, modifique a camada de saída da rede para gerar 2 logitos no lugar de apenas 1 logito. Agora, para converter os logitos em probabilidade, é necessário utilizar a função Softmax, que é dada pela equação:

A função Softmax é usada para prever mais de 2 classes, mas também pode ser usada para o nosso caso de 2 classes. Quando existem N classes, utiliza-se N logitos na saída da rede neural. O Softmax converte estes n logitos em N probabilidades de modo que a soma destas probabilidades é sempre igual a 1, não importando os valores dos logitos que podem assumir quaisquer valores, negativos ou positivos.

O valor da Loss para o caso de N classes e M amostras no batch, é dado por fórmulas especificadas no material.

✓ V.3.a) Repita o exercício V.1.a) porém agora utilizando a equação acima.

```

1 class OneHotMLPClasses(nn.Module):
2     def __init__(self, vocab_size, num_classes):
3         super(OneHotMLPClasses, self).__init__()
4         self.fc1 = nn.Linear(vocab_size + 1, 200)
5         self.fc2 = nn.Linear(200, num_classes) # Duas saídas para as classes positiva e negativa
6         self.relu = nn.ReLU()
7
8     def forward(self, x):
9         o = self.fc1(x.float())
10        o = self.relu(o)
11        return self.fc2(o)

```

```

1 # Suponha que você já tenha definido train_data, vocab_size e OneHotMLP
2
3 train_loader = DataLoader(train_data,
4                             batch_size=128,
5                             shuffle=True,
6                             num_workers=0,
7                             pin_memory=True)
8
9 # Instanciação do modelo
10 model = OneHotMLPClasses(vocab_size,2)
11 model = model.to(device)
12
13 # Obtenha o primeiro batch do train_loader
14 inputs, targets = next(iter(train_loader))
15
16 # Mova os inputs e targets para o dispositivo onde o modelo está
17 inputs = inputs.to(device)
18 targets = targets.to(device)
19
20 # Passe os inputs pelo modelo
21 # Isso retorna os logits, que são os valores brutos de saída do modelo.
22 # Como o modelo ainda não foi treinado, esses valores são inicialmente aleatórios.
23 logits = model(inputs)
24
25 # Calculate the softmax manually
26 exp_logits = torch.exp(logits)
27 sum_exp_logits = torch.sum(exp_logits, dim=1, keepdim=True)
28 probabilities = exp_logits / sum_exp_logits
29
30
31 #print(f'targets {targets}')
32 #print(f'logits {logits}')
33 #print(f'probabilities {probabilities}')
34
35
36 # Calcule a entropia cruzada manualmente
37 # A fórmula é:  $-(y * \log(p) + (1 - y) * \log(1 - p))$ 
38 # onde y é a classe real (0 ou 1) e p é a probabilidade prevista
39 # loss =  $-(targets * \log(probabilities) + (1 - targets) * \log(1 - probabilities)).mean()$ 
40
41
42
43 # Convert targets to one-hot encoding
44 targets_one_hot = torch.nn.functional.one_hot(targets)
45
46 # Calculate the cross-entropy loss manually
47 # torch.nn.functional.one_hot(targets) converte targets para a codificação one-hot.
48 # A fórmula  $-(targets\_one\_hot * \log(probabilities)).sum() / targets.size(0)$ 
49 # calcula a entropia cruzada multiclasse, que é a soma negativa das probabilidades
50 # logarítmicas previstas para as classes verdadeiras, dividida pelo número de amostras.
51
52 # note que isso pressupõe que targets é um tensor de rótulos de classe e que probabilities
53 # é um tensor de probabilidades previstas que soma 1 ao longo da dimensão das classes.
54 loss =  $-(targets\_one\_hot * \log(probabilities)).sum() / targets.size(0)$ 
55
56
57 print(f'loss {loss.item()}')
58
59
60 loss 0.6964499950408936

```

- ✓ **V.3.b)** Modifique a camada de saída da rede para 2 logits e utilize a função Softmax para converter os logits em probabilidades. Repita o exercício V.1.b)

```

1 # Suponha que você já tenha definido train_data, vocab_size e OneHotMLP
2
3 train_loader = DataLoader(train_data,
4                             batch_size=128,
5                             shuffle=True,
6                             num_workers=0,
7                             pin_memory=True)
8
9 # Instanciação do modelo
10 model = OneHotMLPClasses(vocab_size,2)
11 model = model.to(device)
12
13 # Obtenha o primeiro batch do train_loader
14 inputs, targets = next(iter(train_loader))
15
16 # Mova os inputs e targets para o dispositivo onde o modelo está
17 inputs = inputs.to(device)
18 targets = targets.to(device)
19
20 # Passe os inputs pelo modelo
21 # Isso retorna os logits, que são os valores brutos de saída do modelo.
22 # Como o modelo ainda não foi treinado, esses valores são inicialmente aleatórios.
23 logits = model(inputs)
24
25 # A função softmax mapeia probabilidades
26 probabilities = torch.softmax(logits, dim=1)
27
28 #print(f'targets {targets}')
29 #print(f'logits {logits}')
30 #print(f'probabilities {probabilities}')
31
32
33 # Calcule a entropia cruzada manualmente
34 # A fórmula é: - (y * log(p) + (1 - y) * log(1 - p))
35 # onde y é a classe real (0 ou 1) e p é a probabilidade prevista
36 # loss = -(targets * torch.log(probabilities) + (1 - targets) * torch.log(1 - probabilities)).mean()
37
38
39
40 # Convert targets to one-hot encoding
41 targets_one_hot = torch.nn.functional.one_hot(targets)
42
43 # Calculate the cross-entropy loss manually
44 # torch.nn.functional.one_hot(targets) converte targets para a codificação one-hot.
45 # A fórmula -(targets_one_hot * torch.log(probabilities)).sum() / targets.size(0)
46 # calcula a entropia cruzada multiclasse, que é a soma negativa das probabilidades
47 # logarítmicas previstas para as classes verdadeiras, dividida pelo número de amostras.
48
49 # note que isso pressupõe que targets é um tensor de rótulos de classe e que probabilities
50 # é um tensor de probabilidades previstas que soma 1 ao longo da dimensão das classes.
51 loss = -(targets_one_hot * torch.log(probabilities)).sum() / targets.size(0)
52
53
54 print(f'loss {loss.item()}')
55
56
57 loss 0.6906419992446899

```

✓ **V.3.c)** Utilize agora a função `nn.CrossEntropyLoss` para calcular a Loss e verifique se os resultados são os mesmos que anteriormente.

```

1 # Suponha que você já tenha definido train_data, vocab_size e OneHotMLP
2
3 train_loader = DataLoader(train_data,
4                             batch_size=128,
5                             shuffle=True,
6                             num_workers=0,
7                             pin_memory=True)
8
9 # Instanciação do modelo
10 model = OneHotMLPClasses(vocab_size,2)
11 model = model.to(device)
12
13 # Obtenha o primeiro batch do train_loader
14 inputs, targets = next(iter(train_loader))
15
16 # Mova os inputs e targets para o dispositivo onde o modelo está
17 inputs = inputs.to(device)
18 targets = targets.to(device)
19
20 # Passe os inputs pelo modelo
21 # Isso retorna os logits, que são os valores brutos de saída do modelo.
22 # Como o modelo ainda não foi treinado, esses valores são inicialmente aleatórios.
23 logits = model(inputs)
24
25 # A função softmax mapeia probabilidades
26 probabilities = torch.softmax(logits, dim=1)
27
28 # Instantiate the loss function
29 criterion = nn.CrossEntropyLoss()
30
31 # Calculate the loss
32 loss = criterion(logits, targets)
33
34 print(f'loss {loss.item()}')
35
    loss 0.6907879710197449

```

- ✓ **V.3.d)** Modifique as seções V e VI para que o notebook funcione com a saída da rede com 2 logits. Há necessidade de alterar o laço de treinamento e o laço de cálculo da acurácia.

```

1 def evaluate(model, test_loader):
2     ## evaluation
3     model.eval()
4
5     with torch.no_grad():
6         correct = 0
7         total = 0
8         for inputs, labels in test_loader:
9             inputs = inputs.to(device)
10            labels = labels.to(device)
11            outputs = model(inputs)
12
13            if outputs.shape[1] == 1: # gera 1 logito
14                predicted = torch.round(torch.sigmoid(outputs)).view(-1)
15            else: # gera mais de um logito, um para cada classe
16                predicted = torch.argmax(outputs, dim=1).view(-1)
17            total += labels.size(0)
18            correct += (predicted == labels.view(-1)).sum().item()
19
20     test_accuracy = 100 * correct / total
21
22     return test_accuracy
23

```

```

1 def validar_correcao_loss_esperada_xentropy(model, criterion, train_loader):
2
3     # Forward pass com o modelo inicializado
4     # Obtenha o primeiro batch do train_loader
5     inputs, targets = next(iter(train_loader))
6
7     # Mova os inputs e targets para o dispositivo onde o modelo está
8     inputs = inputs.to(device)
9     targets = targets.to(device).view(-1)
10
11     initial_outputs = model(inputs) # Inputs devem ser do DataLoader de treinamento
12
13
14     if initial_outputs.shape[1] == 1: # gera 1 logito
15         loss = criterion(initial_outputs.view(-1), targets.view(-1).float())
16     else: # gera mais de um logito, um para cada classe
17         loss = criterion(nn.functional.softmax(initial_outputs, dim=1), targets)
18
19     # Valor esperado da Loss
20     valor_esperado = 0.6931
21
22     diff = abs(loss.item()-valor_esperado)
23     # Verifique se a Loss inicial está próxima do valor esperado
24     if diff < 0.01: # Considerando igualdade até 2 casas decimais
25         print(f"A Loss inicial de cross entropy {loss} está correta! Diferença de {diff:.4f} (arredondamentos)")
26         return True
27     else:
28         print(f"A Loss inicial de cross entropy não corresponde ao valor esperado. Valor atual: {loss.item():.4f}. Valor esperado por")
29         return False
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
263
```

✓ VI.1 Exercícios:

✓ VI.1.a) Calcule o número de amostras que está sendo considerado na seção de avaliação.

```
1 print(f'0 total de amostras avaliadas em teste é: {len(test_loader.dataset)} em {len(test_loader)} batches')  
0 total de amostras avaliadas em teste é: 25000 em 3125 batches
```

✓ VI.1.b) Explique o que faz os comandos `model.eval()` e `with torch.no_grad()`.

R.:

1. **`model.eval()`:** Quando você chama `model.eval()`, você está essencialmente definindo o modelo (`nn.Module`) para o modo de avaliação. Isso tem efeitos em certas camadas do seu modelo, como Dropout e BatchNorm, que se comportam de maneira diferente durante o treinamento e durante a avaliação. Por exemplo, durante o treinamento, a camada Dropout irá aleatoriamente zerar algumas das entradas, mas durante a avaliação (ou seja, quando o modelo está em modo `.eval()`), a camada Dropout não alterará suas entradas. Da mesma forma, a camada BatchNorm usará estatísticas de execução durante o treinamento, mas usará estatísticas acumuladas durante a avaliação. Embora o modelo possa não possuir essas camadas, há sempre a possibilidade de ele ser evoluído. Outra motivação é a legibilidade do código: deixa claro para qualquer pessoa que esteja lendo o seu código que essa parte do código está fazendo a avaliação do modelo, não o treinamento. Tem também a compatibilidade com outras bibliotecas de aprendizado profundo ou funções do PyTorch que podem ser afetadas com essa informação do modelo.
2. **`with torch.no_grad()`:** Em PyTorch, cada operação em tensores que têm `requires_grad=True` irá criar um histórico de computação que permite calcular gradientes usando backpropagation. No entanto, durante a avaliação do modelo, você geralmente não precisa de gradientes, porque você não está atualizando os pesos do modelo. O uso de `with torch.no_grad()`: desativa a criação desse histórico de computação, o que pode reduzir o uso de memória e acelerar os cálculos. Isso é especialmente útil durante a avaliação do modelo, quando você normalmente só precisa passar os dados através do modelo e calcular a perda ou as métricas de avaliação, sem precisar atualizar os pesos do modelo.

✓ VI.1.c) Existe uma forma mais simples de calcular a classe predita na linha 11, sem a necessidade de usar a função `torch.sigmoid`?

R.: Sim, existe uma maneira mais simples de calcular a classe predita sem a necessidade de usar a função `torch.sigmoid`. A função `torch.sigmoid` é usada para mapear os valores de saída do modelo para o intervalo entre 0 e 1, que pode ser interpretado como a probabilidade da classe positiva. No entanto, se você está apenas interessado na classe predita e não na probabilidade, você pode simplesmente usar a função `torch.round` diretamente nos valores de saída do modelo.

Aqui está como você pode fazer isso:

```
predicted = torch.round(outputs).view(-1)
```

Neste caso, os valores de saída do modelo que são maiores que 0 serão arredondados para 1 (classe positiva) e os valores que são menores ou iguais a 0 serão arredondados para 0 (classe negativa).

No entanto, é importante notar que esta abordagem só é válida se você estiver usando uma função de perda que já inclua a função sigmoid, como a Binary Cross-Entropy with Logits Loss (BCEWithLogitsLoss) no PyTorch. Se você estiver usando uma função de perda que não inclui a função sigmoid, como a Binary Cross-Entropy Loss (BCELoss), você ainda precisará aplicar a função sigmoid aos valores de saída do modelo antes de arredondá-los.

✓ VI.2 Exercícios:

Perplexidade como métrica de avaliação

Em teoria da informação, a perplexidade (PPL) é dada por

$$PPL = e^{CE}$$

onde CE é a Cross Entropy, que utilizamos na Loss do treinamento. A base e utilizada para a exponenciação deve ser compatível com a base utilizado no logaritmo da cross entropia. Como utilizamos logaritmo natural para a entropia cruzada, devemos aqui usar o e. Se a entropia cruzada usasse a base 2, a perplexidade seria 2 elevado à entropia cruzada.

Eu, particularmente gosto de usar a perplexidade em vez de usar a entropia cruzada pelo motivo que ficará explícito nos exercícios a seguir:

- ✓ **VI.2.a)** Utilizando a resposta do exercício V.1.a, que é a Loss teórica de um modelo aleatório de 2 classes, qual é o valor da perplexidade?

```
1 np.log(2)
```

```
0.6931471805599453
```

```
1 print(f'O valor é {round(math.e**np.log(2),3)}')
```

```
0 valor é 2.0
```

- ✓ **VI.2.b)** E se o modelo agora fosse para classificar a amostra em N classes, qual seria o valor da perplexidade para o caso aleatório?

R.: Seria N.

A perplexidade é uma medida comumente usada para avaliar modelos de linguagem. Ela é definida como a potência inversa da probabilidade média do modelo de prever a classe correta. Em outras palavras, a perplexidade de um modelo é o número de escolhas igualmente prováveis que ele efetivamente tem ao fazer uma previsão.

Se um modelo está fazendo previsões aleatórias para um problema de classificação com N classes, então a probabilidade de prever a classe correta para qualquer amostra é $1/N$. Portanto, a perplexidade para o caso aleatório seria N.

Aqui está a justificativa matemática para isso:

A perplexidade (PPL) é definida como a exponencial da entropia cruzada (CE), que é uma medida da diferença entre duas distribuições de probabilidade. A fórmula da perplexidade é:

$$PPL = e^{CE}$$

A entropia cruzada para o caso aleatório é:

$$CE = -\sum (1/N) \log(1/N) = \log N$$

Portanto, a perplexidade para o caso aleatório é:

$$PPL = e^{\log N} = N$$

Isso significa que, para um modelo que faz previsões aleatórias para um problema de classificação com N classes, a perplexidade é N. Em outras palavras, o modelo tem efetivamente N escolhas igualmente prováveis ao fazer uma previsão.

- ✓ **VI.2.c)** Qual é o valor da perplexidade quando o modelo acerta todas as classes com 100% de probabilidade?

Se você respondeu corretamente as 3 questões acima, já é possível entender que a perplexidade é muito mais fácil de entender o seu significado do que o valor da Loss como entropia cruzada.

R.: Se um modelo de classificação é capaz de prever a classe correta para todas as amostras com 100% de probabilidade, então a perplexidade do modelo é 1.

Aqui está a justificativa matemática.

Se o modelo está prevendo a classe correta com 100% de probabilidade, então a distribuição de probabilidade prevista pelo modelo é a mesma que a distribuição de probabilidade verdadeira. Nesse caso, a entropia cruzada é 0, porque a entropia cruzada é mínima quando as duas distribuições $p(x)$ e $q(x)$ (calculada pelo modelo) são iguais.

A entropia cruzada é definida como:

$$CE(p, q) = -\sum p(x) \log(q(x))$$

Nesse caso, para a classe correta, temos $p(x) = 1$ e $q(x) = 1$. Portanto, o termo correspondente na soma da entropia cruzada é $-1 * \log(1) = 0$, porque o logaritmo de 1 é 0. Para todas as outras classes, temos $p(x) = 0$. Portanto, os termos correspondentes na soma da entropia cruzada são $0 * \log(q(x)) = 0$, porque qualquer número multiplicado por 0 é 0. Portanto, a entropia cruzada é a soma de muitos termos 0, o que resulta em 0.

Portanto, a perplexidade quando o modelo acerta todas as classes com 100% de probabilidade é:

$$PPL = CE^0 = 1$$

Isso significa que, para um modelo perfeito que acerta todas as classes com 100% de probabilidade, a perplexidade é 1. Em outras palavras, o modelo tem efetivamente uma única escolha ao fazer uma previsão.

- ✓ **VI.3 Exercícios:**

- ✓ **VI.3.a)** Modifique o código da seção VI - Avaliação, para que além de calcular a acurácia, calcule também a perplexidade. lembrar que $PPL = \text{torch.exp}(CE)$. Assim, será necessário calcular a entropia cruzada, como feito no laço de treinamento.

```
1 def evaluate(model, test_loader, criterion):
2     ## evaluation
3     model.eval()
4
5     with torch.no_grad():
6         correct = 0
7         total = len(test_loader.dataset)
8         total_loss = 0
9         for inputs, labels in test_loader:
10             inputs = inputs.to(device)
11             labels = labels.to(device)
12             outputs = model(inputs)
13
14             if outputs.shape[1] == 1: # gera 1 logito
15                 predicted = torch.round(torch.sigmoid(outputs)).view(-1)
16                 loss = criterion(outputs.view(-1), labels.view(-1).float())
17             else: # gera mais de um logito, um para cada classe
18                 predicted = torch.argmax(outputs, dim=1).view(-1)
19                 loss = criterion(nn.functional.softmax(outputs, dim=1), labels)
20             total_loss += loss.item()
21             correct += (predicted == labels.view(-1)).sum().item()
22
23     test_accuracy = 100 * correct / total
24     test_loss = total_loss / len(test_loader)
25     test_perplexity = torch.exp(torch.tensor(test_loss))
26     return test_accuracy, test_perplexity
27
```

✓ **VI.4 Último exercício:**

Observando Overfitting

Dizemos que o treinamento está overfitting quando treina-se tanto nas mesmas amostras, de modo quase a decorá-lo e quando o modelo fizer a predição em um outro conjunto de amostras, ele não consegue "generalizar" o conhecimento aprendido no treinamento. Uma forma de detectar se o treinamento está entrando no overfitting é calcular, durante o laço de treinamento, tanto a loss de minimização no conjunto de treino, porém ao final de cada época, calcular a loss ou alguma métrica relacionada ao conjunto de teste ou validação.

✓ **VI.4.a)** Modifique o laço de treinamento

para incorporar também o cálculo da avaliação ao final de cada época. Aproveite para reportar também a perplexidade, tanto do treinamento como da avaliação (observe que será mais fácil de interpretar). Essa é a forma usual de se fazer o treinamento, monitorando se o modelo não entra em overfitting. Por fim, como o dataset tem muitas amostras, ele é demorado de entrar em overfitting. Para ficar mais evidente, diminua novamente o número de amostras do dataset de treino de 25 mil para 1 mil amostras e aumente o número de épocas para ilustrar o caso do overfitting, em que a perplexidade de treinamento continua caindo, porém a perplexidade no conjunto de teste começa a aumentar.


```
1 def train_model(model, criterion, optimizer, num_epochs, train_loader, test_loader, verbose:bool=True):
2
3     if validar_correcao_loss_esperada_xentropy(model, criterion, train_loader):
4         # Training loop
5         total_data = len(train_loader.dataset)
6         results = []
```