

Lab 8 - BCC406

REDES NEURAIS E APRENDIZAGEM EM PROFUNDIDADE

LSTM e Transformers

Prof. Eduardo e Prof. Pedro

Objetivos:

- Parte I : Uso de LSTM com uma camada
- Parte II : Uso de LSTM com duas camadas
- Parte III : Uso de LSTM com duas camadas bidirecionais
- Parte IV: Uso de Transformers

Data da entrega : 01/04

- Complete o código (marcado com `ToDo`) e quando requisitado, escreva textos diretamente nos notebooks. Onde tiver *None*, substitua pelo seu código.
- Execute todo notebook e salve tudo em um PDF **nomeado** como "NomeSobrenome-Lab.pdf"
- Envie o PDF via google [FORM](#)

Este notebook é baseado em tensorflow e Keras.

✓ Representações de texto (*Bag-of-Words* vs. *Word Embeddings*) em NLP

Neste exercício prático, vamos explorar duas abordagens diferentes para representar textos em português e treinar um modelo de classificação de texto simples. Usaremos um dataset desafiador de NLP em português – por exemplo, *resenhas de filmes*.

classificação de texto simples. Usaremos um dataset desafiador de NLP em português – por exemplo, [resenhas de filmes](#) traduzidas para PT-BR, rotuladas como *positivas* ou *negativas*. O [dataset](#) contempla o review dado pelos usuários e o sentimento daquele review (positivo/negativo) com relação ao filme. O objetivo é, dado um review (ou resenha) em português, classificar o texto como positivo ou negativo.

O exercício será dividido em duas etapas principais:

1. **LSTM + MLP:** Uso de LSTM com uma única camada e uma MLP para classificação.
2. **LSTMx2 + MLP:** Uso de duas camadas de LSTM e uma MLP para classificação.
3. **LSTMx2B + MLP:** Uso de duas camadas de LSTM bidirecionais e uma MLP para classificação.
4. **Transformers + MLP:** Uso do modelo Transformer, especificamente o BERTimbau, para realizar a classificação.

Para todas as quatro etapas, você pode usar (ou não) uma camada de **embedding** para converter palavras em vetores densos de dimensões menores.

✓ Importando as bibliotecas

Aqui, faremos a importação de todas as bibliotecas que serão usadas nesta prática.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import pandas as pd
4
5 from sklearn.decomposition import PCA
6 from sklearn.feature_extraction.text import CountVectorizer
7 from sklearn.manifold import TSNE
8 from sklearn.model_selection import train_test_split
9 from tensorflow.keras.models import Sequential
10 from tensorflow.keras.layers import Dense, LSTM, Bidirectional
11 from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D
12 from tensorflow.keras.preprocessing.sequence import pad_sequences
13 from tensorflow.keras.preprocessing.text import Tokenizer
14
15 . . .
```

```
15 import torch
16 from transformers import BertTokenizer
17 from transformers import BertForSequenceClassification
18 from transformers import DataCollatorWithPadding
19 from transformers import Trainer
20 from transformers import TrainingArguments
```

✓ Preparando os dados

Aqui assumiremos um dataset de **resenhas de filmes em português**, já categorizadas como sendo de sentimento **positivo** ou **negativo**.

O dataset está disponível em um arquivo CSV, [na pasta da prática](#). Os passos a seguir demonstram como carregar e inspecionar os dados.

O primeiro passo é fazer o download do arquivo.

```
1 !gdown 1KVIxGF6AVD6i43JPT0DBIZzYKJrBMoE5
```

```
Downloading...
```

```
From (original): https://drive.google.com/uc?id=1KVIxGF6AVD6i43JPT0DBIZzYKJrBMoE5
```

```
From (redirected): https://drive.google.com/uc?id=1KVIxGF6AVD6i43JPT0DBIZzYKJrBMoE5&confirm=t&uuid=ee667479-fba4-4e4
```

```
To: /content/imdb-reviews-pt-br.csv
```

```
100% 127M/127M [00:02<00:00, 50.5MB/s]
```

Fazer a leitura do arquivo das resenhas com o pacote Pandas.

```
1 df = pd.read_csv('imdb-reviews-pt-br.csv')
```

Inspecionando os dados.

```
1 # Verificar as primeiras linhas do dataset para entender sua estrutura
```

```
2 print(f"Número de exemplos: {len(df)}")
3 print(df.head(5))
```

Número de exemplos: 49459

	id	text_en \
0	1	Once again Mr. Costner has dragged out a movie...
1	2	This is an example of why the majority of acti...
2	3	First of all I hate those moronic rappers, who...
3	4	Not even the Beatles could write songs everyon...
4	5	Brass pictures movies is not a fitting word fo...

		text_pt	sentiment
0		Mais uma vez, o Sr. Costner arrumou um filme p...	neg
1		Este é um exemplo do motivo pelo qual a maiori...	neg
2		Primeiro de tudo eu odeio esses raps imbecis, ...	neg
3		Nem mesmo os Beatles puderam escrever músicas ...	neg
4		Filmes de fotos de latão não é uma palavra apr...	neg

Explicação: O código acima lê o arquivo CSV contendo as resenhas. Substitua 'imdb-reviews-pt-br.csv' pelo caminho adequado do seu dataset. Usamos `df.head(5)` para ver as primeiras 5 entradas e inspecionar as colunas. Provavelmente, o dataset terá uma coluna para o texto da resenha (por exemplo, `review_pt` ou `texto`) e outra para o rótulo de sentimento (por exemplo, `sentiment` indicando *positivo/negativo*).

O próximo passo é extrair as colunas de texto e rótulo para listas (ou arrays) separados, o que facilitará o manuseio posteriormente.

```
1 texts = df['text_pt'].astype(str).values # convertendo para string por segurança
2 labels = df['sentiment'].map({'neg': 0, 'pos': 1}).values
```

Fazendo uma análise do que foi carregado.

```
1 print("Total de textos:", len(texts))
2 print("Exemplo de texto:", texts[0][:100], "...") # imprime começo do primeiro texto
3 print("Rótulo desse texto:", labels[0])
```

```
Total de textos: 49459
```

```
Exemplo de texto: Mais uma vez, o Sr. Costner arrumou um filme por muito mais tempo do que o necessário. Além das te  
Rótulo desse texto: 0
```

Nota: Caso seu dataset tenha rótulos como "positivo"/"negativo" ou "pos"/"neg", converta-os para valores numéricos (e.g., 1 para positivo, 0 para negativo) conforme mostrado no comentário acima, pois isso facilita o treinamento do modelo.

Para simplificar o exercício e reduzir tempo de processamento (deixando-o *leve*), podemos **opcionalmente** trabalhar com uma amostra menor do dataset. Por exemplo, usar apenas 10.000 exemplos se o conjunto completo for muito grande.

```
1 # OPCIONAL: usar somente uma parte dos dados para treinamento mais rápido (por exemplo, 10000 primeiras linhas)  
2 df = df.sample(10000, random_state=42) # amostra aleatória de 10000 exemplos
```

Agora que os dados estão carregados e prontos, vamos iniciar a **Parte 1: Bag-of-Words + MLP**.

```
1 MAX_WORDS = 5000  
2 EMBEDDING_DIM = 50  
3 MAX_LEN = 100
```

```
1 tokenizer = Tokenizer(num_words=MAX_WORDS, oov_token="<OOV>")  
2 tokenizer.fit_on_texts(texts)  
3 sequences = tokenizer.texts_to_sequences(texts)  
4 X_seq = pad_sequences(sequences, maxlen=MAX_LEN, padding='post', truncating='post')  
5 X_train_seq, X_test_seq, y_train_seq, y_test_seq = train_test_split(  
6     X_seq, labels, test_size=0.2, random_state=42)
```

✓ LSTM + MLP

Nesta parte, vamos implementar uma arquitetura com somente uma única camada (no mínimo 32 unidades) de LSTM e uma MLP para classificação.

```

1 modelo1 = Sequential([
2     Embedding(input_dim=MAX_WORDS, output_dim=EMBEDDING_DIM, input_length=MAX_LEN),
3     LSTM(32, return_sequences=False),
4     Dense(128, activation="relu"),
5     Dense(32, activation="relu"),
6     Dense(1, activation="sigmoid")
7 ])
8
9 modelo1.summary()
10 modelo1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
11
12 modelo1.fit(X_train_seq,
13             y_train_seq,
14             epochs=10,
15             batch_size=64,
16 )
17
18 modelo1.evaluate(X_train_seq, y_train_seq)

```

Model: "sequential_2"


Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	?	0 (unbuilt)
lstm_2 (LSTM)	?	0 (unbuilt)
dense_6 (Dense)	?	0 (unbuilt)
dense_7 (Dense)	?	0 (unbuilt)
dense_8 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

Epoch 1/10

619/619  10s 10ms/step - accuracy: 0.6489 - loss: 0.5916

Epoch 2/10

619/619  5s 8ms/step - accuracy: 0.8414 - loss: 0.3646

Epoch 3/10

```

Epoch 3/10
619/619 ————— 6s 9ms/step - accuracy: 0.8615 - loss: 0.3187
Epoch 4/10
619/619 ————— 10s 8ms/step - accuracy: 0.8771 - loss: 0.2926
Epoch 5/10
619/619 ————— 5s 8ms/step - accuracy: 0.8899 - loss: 0.2636
Epoch 6/10
619/619 ————— 10s 9ms/step - accuracy: 0.9071 - loss: 0.2249
Epoch 7/10
619/619 ————— 10s 9ms/step - accuracy: 0.9219 - loss: 0.1970
Epoch 8/10
619/619 ————— 5s 8ms/step - accuracy: 0.9334 - loss: 0.1751
Epoch 9/10
619/619 ————— 5s 8ms/step - accuracy: 0.9432 - loss: 0.1551
Epoch 10/10
619/619 ————— 5s 8ms/step - accuracy: 0.9518 - loss: 0.1371
1237/1237 ————— 5s 4ms/step - accuracy: 0.9682 - loss: 0.1038
[0.10101693123579025, 0.9692673087120056]

```

✓ LSTMx2 + MLP

Nesta parte, vamos implementar uma arquitetura com duas camadas (no mínimo 32 unidades em cada uma) de LSTM e uma MLP para classificação.

```

1 modelo2 = Sequential([
2     Embedding(input_dim=MAX_WORDS, output_dim=EMBEDDING_DIM, input_length=MAX_LEN),
3     LSTM(64, return_sequences=True),
4     LSTM(32, return_sequences=False),
5     Dense(128, activation="relu"),
6     Dense(32, activation="relu"),
7     Dense(1, activation="sigmoid")
8 ])
9
10 modelo2.summary()
11 modelo2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
12

```

```

12
13 modelo2.fit(X_train_seq,
14             y_train_seq,
15             epochs=10,
16             batch_size=64,
17 )
18
19 modelo2.evaluate(X_train_seq, y_train_seq)

```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	?	0 (unbuilt)
lstm_7 (LSTM)	?	0 (unbuilt)
lstm_8 (LSTM)	?	0 (unbuilt)
dense_15 (Dense)	?	0 (unbuilt)
dense_16 (Dense)	?	0 (unbuilt)
dense_17 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)

Epoch 1/10

619/619 ————— 12s 13ms/step - accuracy: 0.6723 - loss: 0.5876

Epoch 2/10

619/619 ————— 7s 12ms/step - accuracy: 0.8308 - loss: 0.4007

Epoch 3/10

619/619 ————— 7s 11ms/step - accuracy: 0.8540 - loss: 0.3462

Epoch 4/10

619/619 ————— 7s 12ms/step - accuracy: 0.8757 - loss: 0.2975

Epoch 5/10

619/619 ————— 7s 11ms/step - accuracy: 0.8984 - loss: 0.2554

Epoch 6/10

619/619 ————— 10s 11ms/step - accuracy: 0.9122 - loss: 0.2234

Epoch 7/10

619/619 ————— 11s 11ms/step - accuracy: 0.9285 - loss: 0.1903


```

Epoch 8/10
619/619 ————— 7s 12ms/step - accuracy: 0.9419 - loss: 0.1643
Epoch 9/10
619/619 ————— 10s 12ms/step - accuracy: 0.9481 - loss: 0.1484
Epoch 10/10
619/619 ————— 10s 11ms/step - accuracy: 0.9573 - loss: 0.1241
1237/1237 ————— 7s 5ms/step - accuracy: 0.9721 - loss: 0.0977
[0.09570758044719696, 0.9730836153030396]

```

✓ LSTMx2B + MLP

Nesta parte, vamos implementar uma arquitetura com duas camadas (no mínimo 32 unidades em cada uma) de LSTM bidirecional e uma MLP para classificação.

```

1 modelo3 = Sequential([
2     Embedding(input_dim=MAX_WORDS, output_dim=EMBEDDING_DIM, input_length=MAX_LEN),
3     Bidirectional(LSTM(64, return_sequences=True)),
4     Bidirectional(LSTM(32, return_sequences=False)),
5     Dense(128, activation="relu"),
6     Dense(32, activation="relu"),
7     Dense(1, activation="sigmoid")
8 ])
9
10 modelo3.summary()
11 modelo3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
12
13 modelo3.fit(X_train_seq,
14             y_train_seq,
15             epochs=10,
16             batch_size=64,
17 )
18
19 modelo3.evaluate(X_train_seq, y_train_seq)

```

```

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length`

```

```

/usr/local/lib/python3.11/dist-packages/keras/src/layer/core/embedding.py:30: UserWarning: Argument 'input_length'
warnings.warn(

```

Model: "sequential_6"


Layer (type)	Output Shape	Param #
embedding_6 (Embedding)	?	0 (unbuilt)
bidirectional (Bidirectional)	?	0 (unbuilt)
bidirectional_1 (Bidirectional)	?	0 (unbuilt)
dense_18 (Dense)	?	0 (unbuilt)
dense_19 (Dense)	?	0 (unbuilt)
dense_20 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)


Trainable params: 0 (0.00 B)

Non-trainable params: 0 (0.00 B)


Epoch 1/10

619/619  17s 19ms/step - accuracy: 0.6909 - loss: 0.5545


Epoch 2/10

619/619  20s 19ms/step - accuracy: 0.8513 - loss: 0.3408


Epoch 3/10

619/619  21s 19ms/step - accuracy: 0.8824 - loss: 0.2822


Epoch 4/10

619/619  12s 19ms/step - accuracy: 0.9052 - loss: 0.2357


Epoch 5/10

619/619  20s 19ms/step - accuracy: 0.9268 - loss: 0.1951


Epoch 6/10

619/619  20s 18ms/step - accuracy: 0.9478 - loss: 0.1431


Epoch 7/10

619/619  11s 18ms/step - accuracy: 0.9625 - loss: 0.1058


Epoch 8/10


619/619  21s 19ms/step - accuracy: 0.9722 - loss: 0.0833

Epoch 9/10

619/619  12s 19ms/step - accuracy: 0.9798 - loss: 0.0603

Epoch 10/10

619/619  20s 19ms/step - accuracy: 0.9825 - loss: 0.0543

1237/1237  11s 9ms/step - accuracy: 0.9900 - loss: 0.0332

[0.03272402659058571, 0.9903707504272461]

✓ Transformers + MLP

Nesta parte, vamos implementar uma arquitetura utilizando o BERTimbau e uma MLP para classificação. Para utilizar o BERTimbau no seu modelo, siga o tutorial neste [link](#).

```
1 !pip install -q datasets
```

```
1 from datasets import Dataset, DatasetDict
2 import numpy as np
3 from sklearn.metrics import accuracy_score
```

```
1 MODEL = "neuralmind/bert-large-portuguese-cased"
2 BATCH_SIZE= 8
3 NUM_LABELS = 2
```

```
1 # Diminuição da quantidade de itens para conseguir rodar no Colab
2 # Antes o tempo estipulado para treinamento ultrapassava 18 horas
3
4 texts = texts[:500]
5 print(texts.shape)
6
7 labels = labels[:500]
8 print(labels.shape)
```

```
(500,)
(500,)
```

```
1
2 def convert_to_huggingface(train_texts, train_labels, val_texts, val_labels, test_texts, test_labels):
3     train_dataset = Dataset.from_dict({'text': train_texts, 'label': train_labels})
```

```
4     val_dataset = Dataset.from_dict({'text': val_texts, 'label': val_labels})
5     test_dataset = Dataset.from_dict({'text': test_texts, 'label': test_labels})
6     return DatasetDict({"train": train_dataset, "validation": val_dataset, "test": test_dataset})
7
8
9 def tokenize_function(example):
10     return tokenizer(example['text'], padding="max_length", truncation=True, max_length=512)
11
12 def compute_metrics(eval_pred):
13     predictions, labels = eval_pred
14     preds = np.argmax(predictions, axis=1)
15     return {"accuracy": accuracy_score(labels, preds)}
16
17
18 X_temp, X_val, y_temp, y_val = train_test_split(texts, labels, test_size=0.1)
19 X_train, X_test, y_train, y_test = train_test_split(X_temp, y_temp, test_size=0.2)
20
21
22 dataset = convert_to_huggingface(
23     train_texts=list(X_train), train_labels=list(y_train),
24     val_texts=list(X_val), val_labels=list(y_val),
25     test_texts=list(X_test), test_labels=list(y_test)
26 )
27
28 tokenizer = BertTokenizer.from_pretrained(MODEL)
29
30 tokenized_datasets = dataset.map(tokenize_function, batched=True)
31
32 data_collator = DataCollatorWithPadding(tokenizer=tokenizer, padding="max_length", max_length=512)
33
34 train_dataset = tokenized_datasets['train']
35 eval_dataset = tokenized_datasets['validation']
36
37 model = BertForSequenceClassification.from_pretrained(MODEL, num_labels=NUM_LABELS)
```

Mostrar saída oculta

```
1 training_args = TrainingArguments(/
```

```

1 training_args = TrainingArguments(
2     evaluation_strategy = "epoch",
3     learning_rate=2e-5,
4     per_device_train_batch_size=BATCH_SIZE,
5     num_train_epochs=5,
6     weight_decay=0.01,
7     report_to="none"
8 )
9
10 trainer = Trainer(
11     model=model,
12     args=training_args,
13     train_dataset=train_dataset,
14     eval_dataset=eval_dataset,
15     data_collator=data_collator,
16     compute_metrics=compute_metrics
17 )
18
19 trainer.train()
20
21 # ToDo: Seu código aqui para validar a resposta do seu modelo
22 test_dataset = tokenized_datasets['test']
23 results = trainer.evaluate(test_dataset)
24 print(results)

```

/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611: FutureWarning: `evaluation_strategy` is deprecated in favor of `eval_strategy` and will be removed in a future version.

warnings.warn([225/225 10:08, Epoch 5/5]

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.000096	1.000000
2	No log	0.000052	1.000000
3	No log	0.000039	1.000000
4	No log	0.000033	1.000000
5	No log	0.000032	1.000000

 [12/12 00:08]

```
{'eval_loss': 3.240055593778379e-05, 'eval_accuracy': 1.0, 'eval_runtime': 9.0384, 'eval_samples_per_second': 9.957,
```