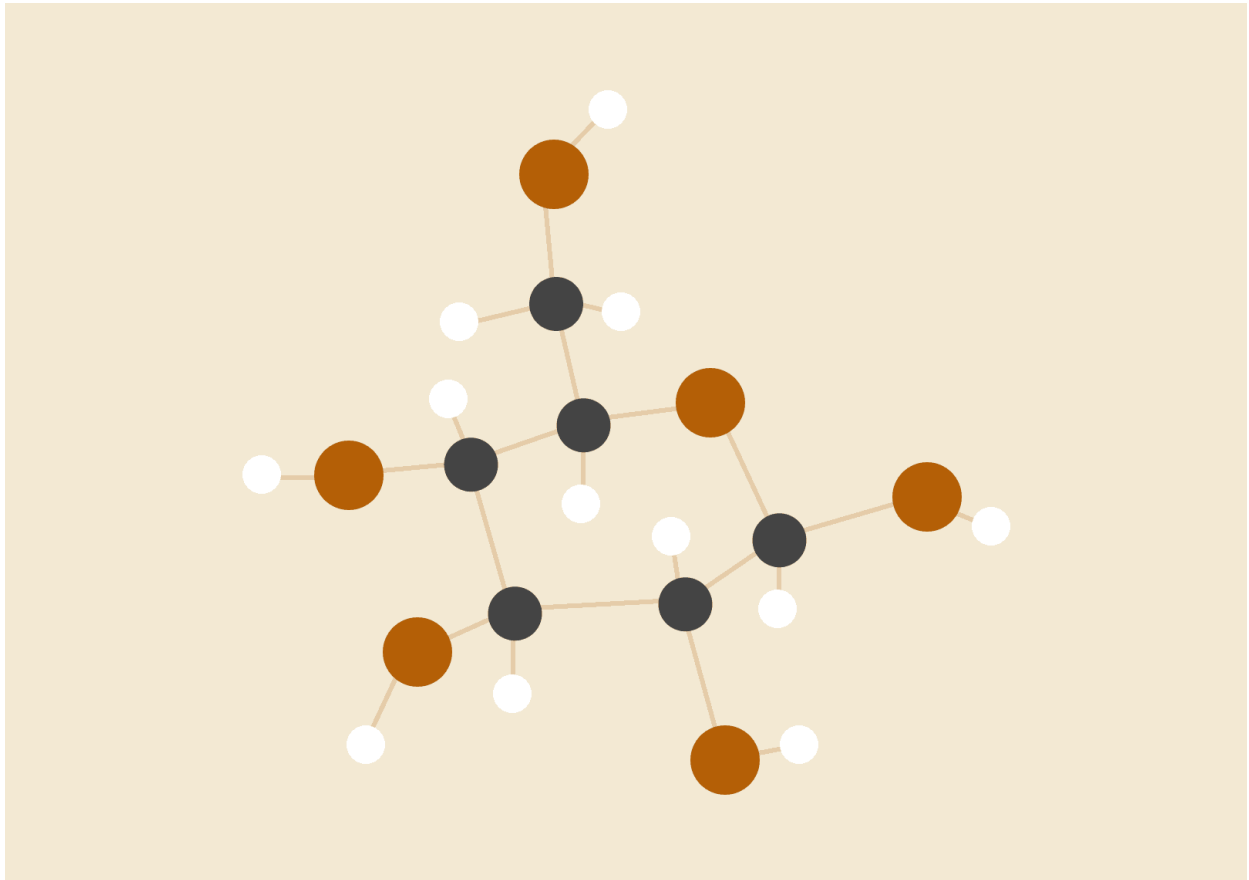


INFORME LABORATORIO 2 DE REDES NEURONALES

Redes Neuronales Recurrentes para procesamiento de lenguaje natural



Cristian Astorga

15 de diciembre de 2022
Ingeniería matemática - USACH

INTRODUCCIÓN

Las redes neuronales recurrentes son una clase de modelos de aprendizaje automático que se utilizan comúnmente en el procesamiento del lenguaje natural. Estos modelos son llamados "recurrentes" porque utilizan conexiones recurrentes entre sus nodos, lo que permite que la red retenga información a largo plazo y procese secuencias de entrada de longitud variable.

Dentro de las redes neuronales recurrentes, una de las arquitecturas más populares es la red LSTM (Long Short-Term Memory), que se caracteriza por su uso de celdas LSTM para controlar el flujo de información a través de la red. Otra arquitectura comúnmente utilizada en el procesamiento del lenguaje natural es la red GRU (Gated Recurrent Unit), que utiliza puertas para controlar el flujo de información a través de la red y olvidar o mantener información anterior.

En este informe, utilizaremos las redes LSTM y GRU y cómo variar sus parámetros afecta en el procesamiento del lenguaje natural.

PROCEDIMIENTO

1. Implementaremos 2 modelos de red recurrente LSTM y GRU
2. utilizaremos una base con aprox 1.1 Millones de reviews entre positivas y negativas sobre hoteles del mundo
3. Haremos un entrenamiento acotado con tan solo 10.000 reviews entre positivas y negativas, iremos variando distintos parámetros de la red como learning rate, épocas de entrenamiento, batch size, etc.
4. Una vez encontremos parámetros buenos para esta cantidad de data, aumentaremos la cantidad de datos hasta encontrar un buen rendimiento
5. Se volverán a modificar parámetros como

DATOS Y PROCESO GENERAL

Usaremos el Dataset presente en el siguiente link: “[515K Hotel Reviews Data in Europe | Kaggle](#)”, luego de cargar los hacemos una exploración inicial:

```
df = pd.read_csv('/content/drive/MyDrive/Data/Hotel_Reviews.csv')
df
```

	Hotel_Address	Additional_Number_of_Scoring	Review_Date	Average_Score	Hotel_Name	Reviewer_Nationality	Negative_Review	Review_Total_Negative_Word_Counts	Total_Number_of_Reviews	Positive_Review
0	Gravesandestraat 55 Oost 1092 AA Amsterdam ...	194	8/3/2017	7.7	Hotel Arena	Russia	I am so angry that i made this post available...	397	1403	Only the park outside of the hotel was beauti...
1	Gravesandestraat 55 Oost 1092 AA Amsterdam ...	194	8/3/2017	7.7	Hotel Arena	Ireland	No Negative	0	1403	No real complaints the hotel was great ...
2	Gravesandestraat 55 Oost 1092 AA Amsterdam ...	194	7/31/2017	7.7	Hotel Arena	Australia	Rooms are nice but for elderly a bit difficul...	42	1403	Location was good and staff were ok It is cut...
3	Gravesandestraat 55 Oost 1092 AA Amsterdam ...	194	7/31/2017	7.7	Hotel Arena	United Kingdom	My room was dirty and I was afraid to walk ba...	210	1403	Great location in nice surroundings the bar a...
4	Gravesandestraat 55 Oost 1092 AA Amsterdam ...	194	7/24/2017	7.7	Hotel Arena	New Zealand	You When I booked with your company on line y...	140	1403	Amazing location and building Romantic setting
...
515733	Wurzbachgasse 21 15 Rudolfsheim F nrlhaus 1150 ...	168	8/30/2015	8.1	Atlantis Hotel Vienna	Kuwait	no trolly or staff to help you take the luggage...	14	2823	location

A lo cual notamos que el dataset en realidad son 515 mil reviews pero que fueron separadas en parte negativa y positiva cada una de ellas, como resultado tenemos datos “No Negative” y “No Positive” entre ellos cuando no había partes negativas o positivas en cada review, por lo cual haremos una limpieza para borrar estos datos.

Además balanceamos las clases, las etiquetamos con 0 para positivos y 1 para negativos, separamos en train y test y guardamos la información en una tupla

```

[7] Negatives = [value for value in df['Negative_Review'].tolist() if value != 'No Negative']
Positives = [value for value in df['Positive_Review'].tolist() if value != 'No Positive']

print( f'Cantidad de Reviews Negativas: {len(Negatives)}, Cantidad de Reviews Positivas: {len(Positives)}')

while len(Positives) > 5000:
    Positives.pop()
while len(Negatives) > 5000:
    Negatives.pop()
print('-----')
print('Luego del Balanceo tenemos:')
print( f'Cantidad de Reviews Negativas: {len(Negatives)}, Cantidad de Reviews Positivas: {len(Positives)}')

train_data_N, test_data_N = train_test_split(Negatives, test_size=0.2, random_state=42)
train_data_P, test_data_P = train_test_split(Positives, test_size=0.2, random_state=42)

labels0 = [0 for value in train_data_P]
labels00 = [0 for value in test_data_P]
labels1 = [1 for value in train_data_N]
labels11 = [1 for value in test_data_P]

train_data = train_data_P + train_data_N
test_data = test_data_P + test_data_N
labels_train = labels0 + labels1
labels_test = labels00 + labels11

train_data = train_data, labels_train
test_data = test_data, labels_test

print('-----')
print('Luego de la separación entre train y test tenemos:')
print( f'Cantidad de Reviews en train: {len(train_data[0])}, Cantidad de Reviews en test: {len(test_data[0])}')

```

Cantidad de Reviews Negativas: 387848, Cantidad de Reviews Positivas: 479792

 Luego del Balanceo tenemos:
 Cantidad de Reviews Negativas: 5000, Cantidad de Reviews Positivas: 5000

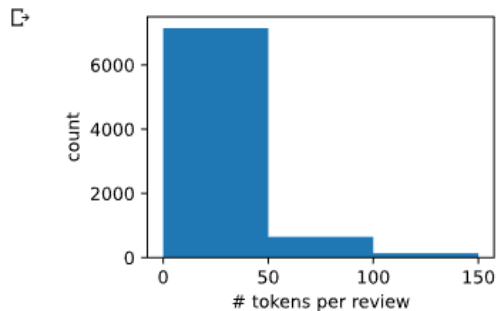
 Luego de la separación entre train y test tenemos:
 Cantidad de Reviews en train: 8000, Cantidad de Reviews en test: 2000

Luego separamos las palabras en “tokens” para su manipulación, un vocabulario donde tendremos todas las palabras que se repitan al menos 5 veces entre todas las reviews y en el gráfico podemos observar la cantidad de palabras por reviews. Con esta información podemos ajustar los parámetros num_steps que es el número de palabras fijo por review (las reviews con menos palabras serán rellenadas con <pad>) y batch_size que es el número de palabras que se analizan por iteración.

Tokenizamos nuestra data, establecemos nuestro vocabulario y eliminamos las palabras que aparecen menos de 5 veces.

```
[ ] train_tokens = d2l.tokenize(train_data[0], token='word')
    test_tokens = d2l.tokenize(test_data[0], token='word')
    vocab = d2l.Vocab(train_tokens, min_freq=5)
```

```
▶ d2l.set_figsize()
  d2l.plt.xlabel('# tokens per review')
  d2l.plt.ylabel('count')
  d2l.plt.hist([len(line) for line in train_tokens], bins=range(0, 200, 50));
```



```
[ ] num_steps=150
    batch_size = 6

    train_features = torch.tensor([d2l.truncate_pad(vocab[line], num_steps, vocab['<pad>']) for line in train_tokens])
    test_features = torch.tensor([d2l.truncate_pad(vocab[line], num_steps, vocab['<pad>']) for line in test_tokens])
    train_iter = d2l.load_array((train_features, torch.tensor(train_data[1])), batch_size)
    test_iter = d2l.load_array((test_features, torch.tensor(test_data[1])), batch_size, is_train=False)
```

En este caso ajustamos 150 palabras máximo por review para no acortar las opiniones y 6 palabras por iteración de procesamiento.

Para comenzar usaremos una red GRU definida a continuación:

```
class BiRNN(nn.Module):
    def __init__(self, vocab_size, embed_size, num_hiddens,
                  num_layers, **kwargs):
        super(BiRNN, self).__init__(**kwargs)
        self.embedding = nn.Embedding(vocab_size, embed_size)
        # Set `bidirectional` to True to get a bidirectional RNN
        self.encoder = nn.GRU(embed_size, num_hiddens, num_layers=num_layers,
                               bidirectional=True)
        self.decoder = nn.Linear(4 * num_hiddens, 2)

    def forward(self, inputs):
        # The shape of `inputs` is (batch size, no. of time steps). Because
        # GRU requires its input's first dimension to be the temporal
        # dimension, the input is transposed before obtaining token
        # representations. The output shape is (no. of time steps, batch size,
        # word vector dimension)
        embeddings = self.embedding(inputs.T)
        self.encoder.flatten_parameters()
        # Returns hidden states of the last hidden layer at different time
        # steps. The shape of `outputs` is (no. of time steps, batch size,
        # 2 * no. of hidden units)
        outputs, _ = self.encoder(embeddings)
        # Concatenate the hidden states at the initial and final time steps as
        # the input of the fully connected layer. Its shape is (batch size,
        # 4 * no. of hidden units)
        encoding = torch.cat((outputs[0], outputs[-1]), dim=1)
        outs = self.decoder(encoding)
        return outs
```

Otros posibles parámetros a modificar son el tamaño del embedding, número de neuronas y número de capas

```
[ ] embed_size, num_hiddens, num_layers, devices = 100, 100, 3, d2l.try_all_gpus()
    net = BiRNN(len(vocab), embed_size, num_hiddens, num_layers)
```

Usaremos el Embedding Glove con dimensión 100:

Embedding

```
▶ glove_embedding = d2l.TokenEmbedding('glove.6b.100d')  
↳ Downloading ../data/glove.6B.100d.zip from http://d2l-data.s3-accelerate.amazonaws.com/glove.6B.100d.zip  
[ ] embeds = glove_embedding[vocab.idx_to_token]  
    embeds.shape  
↳ torch.Size([2604, 100])  
[ ] net.embedding.weight.data.copy_(embeds)  
    net.embedding.weight.requires_grad = False  
[ ] devices  
    [device(type='cuda', index=0)]
```

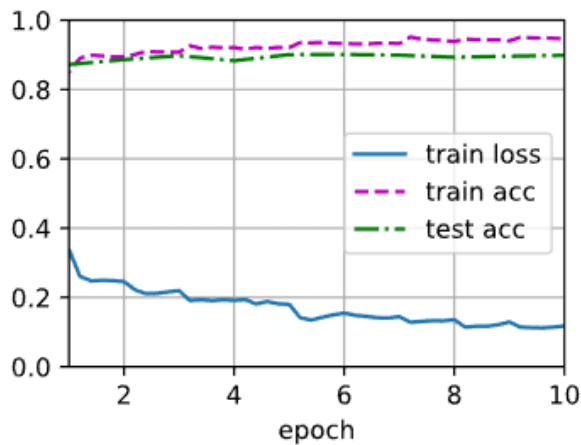
El Entrenamiento debe ser por GPU para que sea en un tiempo razonable.

Luego entrenamos y graficamos el rendimiento de nuestro modelo:

```
# usando toda la data
```

```
lr, num_epochs = 0.003, 10
trainer = torch.optim.Adam(net.parameters(), lr=lr)
loss = nn.CrossEntropyLoss(reduction="none")
d2l.train_ch13(net, train_iter, test_iter, loss, trainer, num_epochs, devices)
```

```
loss 0.118, train acc 0.946, test acc 0.898
913.9 examples/sec on [device(type='cuda', index=0)]
```



Finalmente usamos algunas frases de prueba para probar nuestro modelo, para ello definimos una función de predicción:

```
#@save
def predict_sentiment(net, vocab, sequence):
    """Predict the sentiment of a text sequence."""
    sequence = torch.tensor(vocab[sequence.split()], device=d2l.try_gpu())
    label = torch.argmax(net(sequence.reshape(1, -1)), dim=1)
    return 'Negative' if label == 1 else 'Positive'
```



```
[ ] predict_sentiment(net, vocab, 'this hotel is awesome')
    'Positive'

[ ] predict_sentiment(net, vocab, "I've been in better hotels, but it wasn't that bad either.")
    'Negative'

[ ] predict_sentiment(net, vocab, "the service leaves a lot to be desired")
    'Negative'

[ ] predict_sentiment(net, vocab, "the attention leaves much to be desired, especially the bathroom that was dirty")
    'Negative'

▶ predict_sentiment(net, vocab, "the worst place i have been in my life")
↳ 'Negative'

[ ] predict_sentiment(net, vocab, "I love this hotel")
    'Positive'

[ ] predict_sentiment(net, vocab, "I like this hotel")
    'Negative'
```

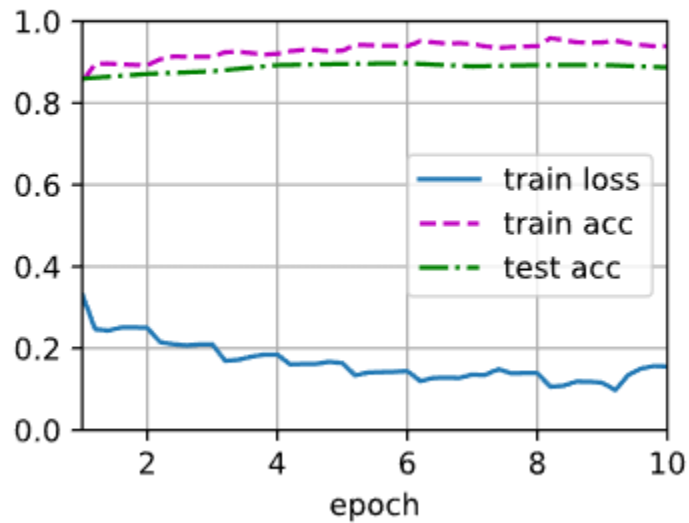
Solamente encontramos un error entre nuestras pruebas, lo malo es que es una frase bastante común la que falló, esto puede ser porque la única palabra que indica si es positiva o negativa la review es “like” la cual puede ser usada en muchos contextos en el inglés.

RESULTADOS

A continuación mostraremos resultados de entrenamiento para diferentes parámetros para la red GRU:

1. **Tipo de red: GRU, cantidad de datos: 10.000**, tamaño del embedding = 100, número de neuronas = 100, **número de capas = 5, batch size = 12**, número de palabras por review = 150, learning rate = 0.003, épocas = 10

```
loss 0.156, train acc 0.939, test acc 0.887
1743.5 examples/sec on [device(type='cuda', index=0)]
```



Aparentemente tenemos un entrenamiento parecido, veamos con las pruebas de texto

```
[43] predict_sentiment(net, vocab, "I've been in better hotels, but it wasn't that bad either.")
      'Negative'
```

```
[44] predict_sentiment(net, vocab, "the service leaves a lot to be desired")
      'Negative'
```

```
[45] predict_sentiment(net, vocab, "the attention leaves much to be desired, especially the bathroom that was dirty")
      'Negative'
```

```
[46] predict_sentiment(net, vocab, "the worst place i have been in my life")
      'Negative'
```

```
[47] predict_sentiment(net, vocab, "I love this hotel")
      'Positive'
```

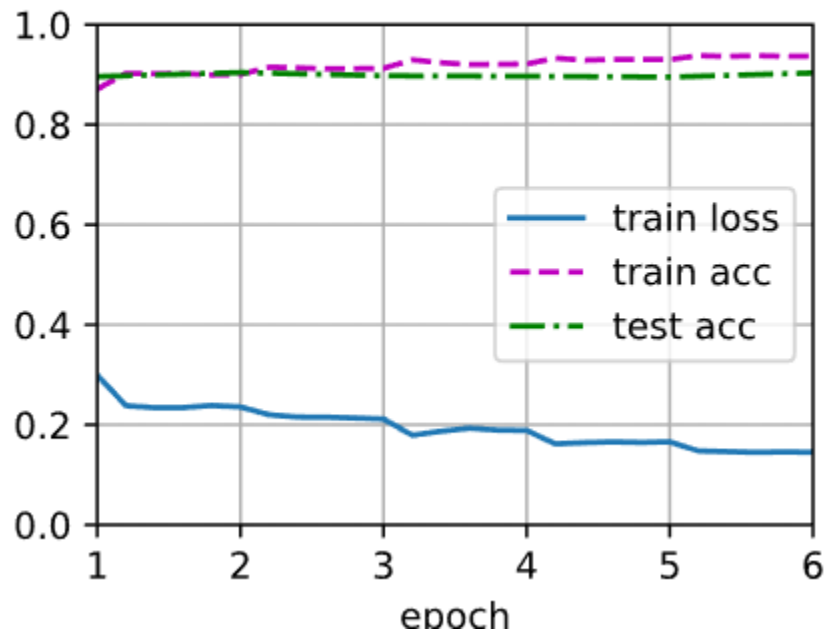
```
[48] predict_sentiment(net, vocab, "I like this hotel")
      'Positive'
```

Esta vez fueron todas correctas

2. Tipo de red: GRU, **cantidad de datos: 30.000**, tamaño del embedding = 100,


número de neuronas = 100, número de capas = 5, **batch size** = 24, número de palabras por review = 150, **learning rate** = 0.001, épocas = 6

loss 0.145, train acc 0.937, test acc 0.903
1871.1 examples/sec on [device(type='cuda', index=0)]



La red no tuvo problemas para detectar las frases anteriores, esta vez comenzaremos a usar otras frases más difíciles:

```

✓  predict_sentiment(net, vocab, 'this hotel is awesome')
  'Positive'

✓ [70] predict_sentiment(net, vocab, "I've been in better hotels, but it wasn't that bad either.")
  'Negative'

✓ [71] predict_sentiment(net, vocab,"the service leaves a lot to be desired")
  'Negative'

✓ [72] predict_sentiment(net, vocab,"the attention leaves much to be desired, especially the bathroom that was dirty")
  'Negative'

✓ [76] predict_sentiment(net,vocab,"I would not return to this hotel even if they paid me")
  'Negative'

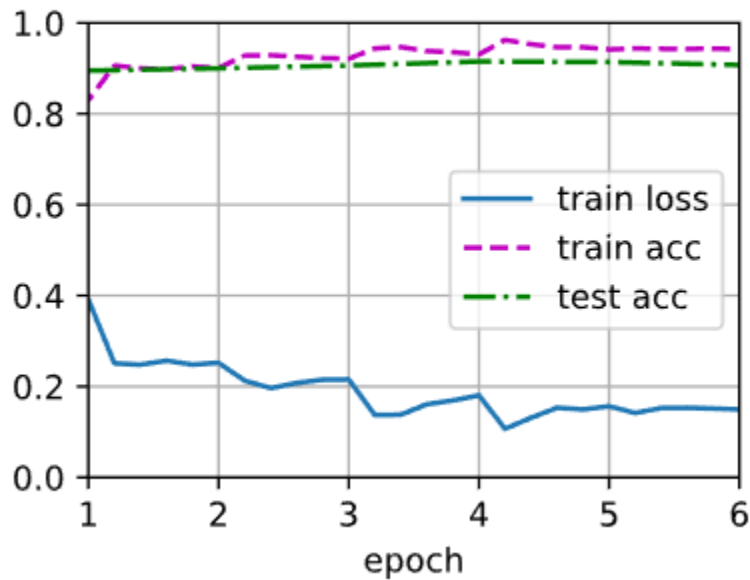
✓ [75] predict_sentiment(net,vocab,"I like this hotel")
  'Positive'

✓ [78] predict_sentiment(net,vocab,"The hotel is too big, I couldn't fully explore it")
  'Negative'

```

3. Tipo de red: GRU, **cantidad de datos: 4.000**, tamaño del embedding = 100, número de neuronas = 100, número de capas = 3, **batch size = 8**, número de palabras por review = 150, **learning rate = 0.003**, épocas = 6

```
loss 0.149, train acc 0.943, test acc 0.907
1172.4 examples/sec on [device(type='cuda', index=0)]
```



```
[120] predict_sentiment(net, vocab, 'this hotel is awesome')
```

```
'Positive'
```

```
[121] predict_sentiment(net, vocab, "I've been in better hotels, but it wasn't that bad either.")
```

```
'Negative'
```

```
[122] predict_sentiment(net, vocab, "the service leaves a lot to be desired")
```

```
'Positive'
```

```
[123] predict_sentiment(net, vocab, "the attention leaves much to be desired, especially the bathroom that was dirty")
```

```
'Negative'
```

```
[124] predict_sentiment(net, vocab, "I would not return to this hotel even if they paid me")
```

```
'Negative'
```

```
[125] predict_sentiment(net, vocab, "I like this hotel")
```

```
'Positive'
```

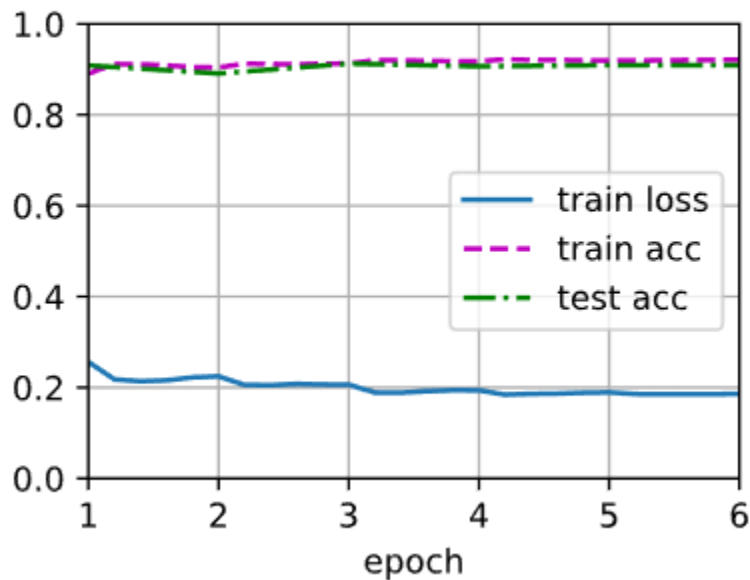
```
[126] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it")
```

```
'Negative'
```

Resultados iguales al anterior, excepto la tercera frase que fue catalogada mal.

4. Tipo de red: GRU, **cantidad de datos: 100.000**, tamaño del embedding = 100, número de neuronas = 100, **número de capas = 4**, **batch size = 26**, número de palabras por review = 150, learning rate = 0.003, épocas = 6

loss 0.186, train acc 0.921, test acc 0.909
2401.9 examples/sec on [device(type='cuda', index=0)]



```
[144] predict_sentiment(net, vocab, 'this hotel is awesome')
      'Positive'

[145] predict_sentiment(net, vocab, "I've been in better hotels, but it wasn't that bad either.")
      'Negative'

[146] predict_sentiment(net, vocab, "the service leaves a lot to be desired")
      'Negative'

[147] predict_sentiment(net, vocab, "the attention leaves much to be desired, especially the bathroom that was dirty")
      'Negative'

[148] predict_sentiment(net, vocab, "I would not return to this hotel even if they paid me")
      'Negative'

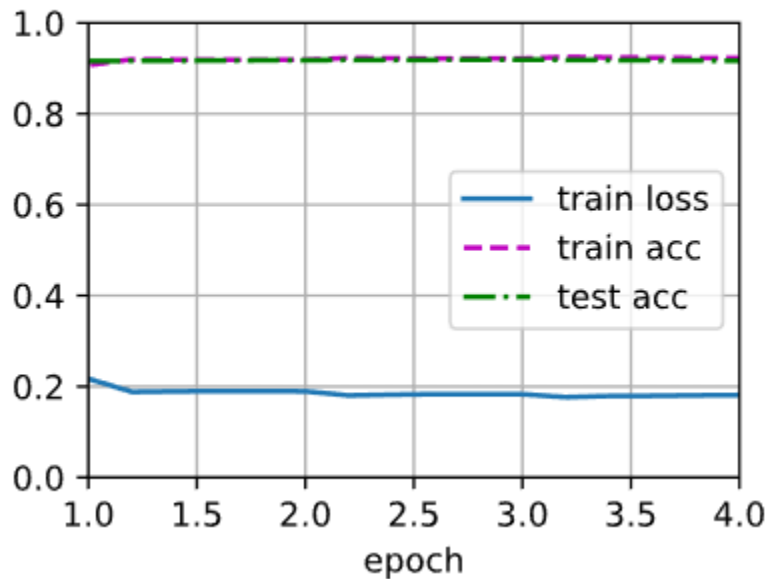
[149] predict_sentiment(net, vocab, "I like this hotel")
      'Positive'

[150] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it")
      'Negative'
```

5. Tipo de red: GRU, **cantidad de datos: 775.696**, tamaño del embedding = 100,

número de neuronas = 100, número de capas = 4, **batch size** = 18, número de palabras por review = 150, learning rate = 0.003, épocas = 6

loss 0.181, train acc 0.923, test acc 0.917
2012.9 examples/sec on [device(type='cuda', index=0)]



```
[200] predict_sentiment(net, vocab, 'this hotel is awesome')
```

```
'Positive'
```

```
[201] predict_sentiment(net, vocab, "I've been in better hotels, but it wasn't that bad either.")
```

```
'Negative'
```

```
▶ predict_sentiment(net, vocab, "the service leaves a lot to be desired")
```

```
↳ 'Negative'
```

```
[203] predict_sentiment(net, vocab, "the attention leaves much to be desired, especially the bathroom that was dirty")
```

```
'Negative'
```

```
[204] predict_sentiment(net, vocab, "I would not return to this hotel even if they paid me")
```

```
'Negative'
```

```
[205] predict_sentiment(net, vocab, "I like this hotel")
```

```
'Positive'
```

```
[206] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it")
```

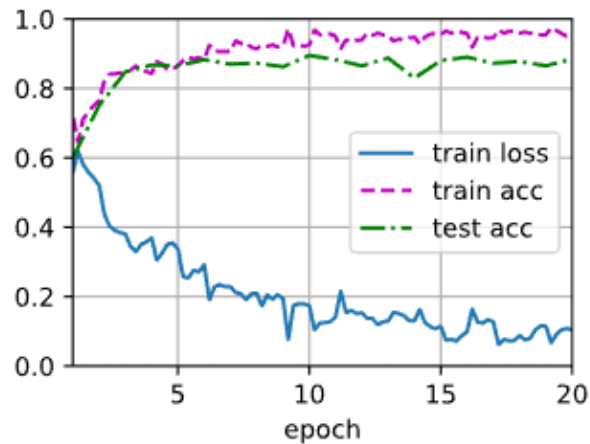
```
'Negative'
```

Ahora comenzaremos a utilizar red LSTM:

6. **Tipo de red: LSTM, cantidad de datos: 2.000**, tamaño del embedding = 100, número de neuronas = 100, número de capas = 4, **batch size = 6**, número de palabras por review = 150, frecuencia mínima por palabra=5, learning rate = 0.003, épocas = 20

```
lr, num_epochs = 0.003, 20  
trainer = torch.optim.Adam(net.parameters(), lr=lr)  
loss = nn.CrossEntropyLoss(reduction="none")  
d2l.train_ch13(net, train_iter, test_iter, loss, trainer, num_epochs, devices)
```

```
loss 0.104, train acc 0.952, test acc 0.885  
534.0 examples/sec on [device(type='cuda', index=0)]
```




```
[21] predict_sentiment(net, vocab, 'this hotel is awesome')
      'Positive'

▶ predict_sentiment(net, vocab, "I've been in better hotels, but it wasn't that bad either.")
      'Negative'

[23] predict_sentiment(net, vocab, "the service leaves a lot to be desired")
      'Negative'

[24] predict_sentiment(net, vocab, "the attention leaves much to be desired, especially the bathroom that was dirty")
      'Negative'

[25] predict_sentiment(net, vocab, "the worst place i have been in my life")
      'Negative'

[26] predict_sentiment(net, vocab, "I love this hotel")
      'Positive'

[27] predict_sentiment(net, vocab, "I like this hotel")
      'Positive'

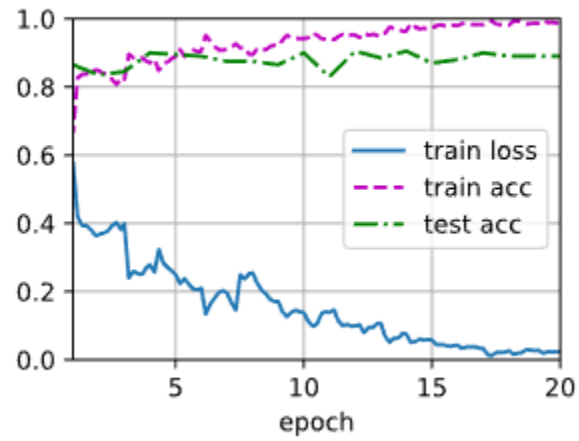
[28] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it")
      'Negative'

[29] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it. I liked")
      'Positive'
```

La red LSTM tiene buenos resultados con tan solo 2.000 datos

7. Tipo de red: LSTM, **cantidad de datos: 1.000**, tamaño del embedding = 100, número de neuronas = 100, **número de capas = 2**, **batch size = 36**, número de palabras por review = 150, frecuencia mínima por palabra=5, learning rate = 0.03, épocas = 20

```
loss 0.022, train acc 0.989, test acc 0.890  
1980.6 examples/sec on [device(type='cuda', index=0)]
```



Sobre entrenamiento por pocos datos y muchas épocas, veamos que tal los resultados de textos:

```
[47] predict_sentiment(net, vocab, 'this hotel is awesome')
      'Positive'

[48] predict_sentiment(net, vocab, "I've been in better hotels, but it wasn't that bad either.")
      'Negative'

[49] predict_sentiment(net, vocab, "the service leaves a lot to be desired")
      'Negative'

[50] predict_sentiment(net, vocab, "the attention leaves much to be desired, especially the bathroom that was dirty")
      'Negative'

[51] predict_sentiment(net, vocab, "the worst place i have been in my life")
      'Negative'

[52] predict_sentiment(net, vocab, "I love this hotel")
      'Positive'

[53] predict_sentiment(net, vocab, "I like this hotel")
      'Negative'

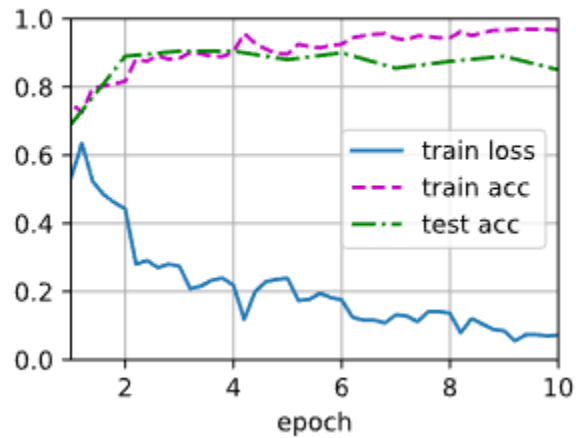
[54] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it")
      'Negative'

[55] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it. I liked")
      'Negative'
```

Clasificó mal la frase “i like this hotel”, la cual es una frase sencilla, puede ser debido a los distintos contextos en que se usa “like” en inglés, intentaremos mejorar el entrenamiento y mantener la cantidad de datos que sabemos puede ser limitada.

8. Tipo de red: LSTM, **cantidad de datos: 1.000**, tamaño del embedding = 100, número de neuronas = 100, número de capas = 2, **batch size = 16**, **número de palabras por review = 50**, **frecuencia mínima por palabra=2**, learning rate = 0.003, **épocas = 10**

```
loss 0.072, train acc 0.966, test acc 0.850
2365.0 examples/sec on [device(type='cuda', index=0)]
```



```
[73] predict_sentiment(net, vocab, 'this hotel is awesome')
```

```
'Positive'
```

```
[74] predict_sentiment(net, vocab, "I've been in better hotels, but it wasn't that bad either.")
```

```
[->] 'Negative'
```

```
[75] predict_sentiment(net, vocab, "the service leaves a lot to be desired")
```

```
[->] 'Negative'
```

```
[76] predict_sentiment(net, vocab, "the attention leaves much to be desired, especially the bathroom that was dirty")
```

```
[->] 'Negative'
```

```
[77] predict_sentiment(net, vocab, "the worst place i have been in my life")
```

```
'Negative'
```

```
[78] predict_sentiment(net, vocab, "I love this hotel")
```

```
'Negative'
```

```
[79] predict_sentiment(net, vocab, "I like this hotel")
```

```
'Negative'
```

```
[80] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it")
```

```
'Negative'
```

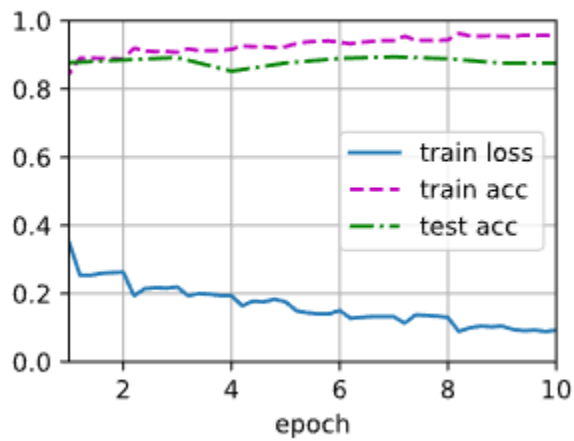
```
[81] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it. I liked")
```

```
'Negative'
```

Mal resultado, es necesario aumentar los datos o los parámetros son malos?, probaremos con más datos solamente

9. Tipo de red: LSTM, **cantidad de datos: 10.000**, tamaño del embedding = 100, número de neuronas = 100, número de capas = 2, **batch size = 16**, **número de palabras por review = 50**, **frecuencia mínima por palabra=2**, learning rate = 0.003, épocas = 10

```
loss 0.093, train acc 0.956, test acc 0.875  
3254.1 examples/sec on [device(type='cuda', index=0)]
```



```
[99] predict_sentiment(net, vocab, 'this hotel is awesome')
      'Positive'

[100] predict_sentiment(net, vocab, "I've been in better hotels, but it wasn't that bad either.")
      'Negative'

[101] predict_sentiment(net, vocab, "the service leaves a lot to be desired")
      'Negative'

[102] predict_sentiment(net, vocab, "the attention leaves much to be desired, especially the bathroom that was dirty")
      'Negative'

[103] predict_sentiment(net, vocab, "the worst place i have been in my life")
      'Negative'

[104] predict_sentiment(net, vocab, "I love this hotel")
      'Positive'

▶ predict_sentiment(net, vocab, "I like this hotel")
🔗 'Positive'

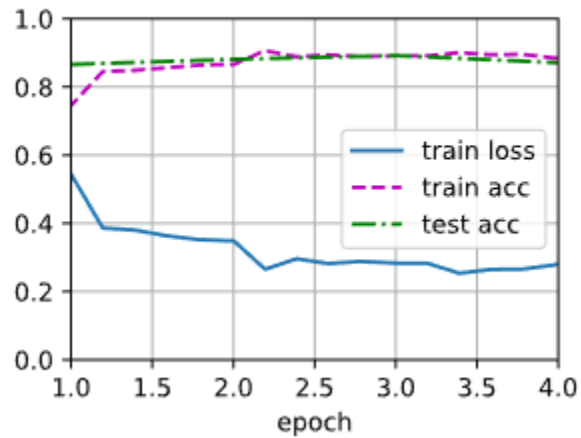
[106] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it")
      'Negative'

[107] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it. I liked")
      'Positive'
```

El resultado mejora al tener más datos

10. Tipo de red: LSTM, **cantidad de datos: 5000**, tamaño del embedding = 100, número de neuronas = 100, número de capas = 2, **batch size = 26**, número de palabras por review = 50, frecuencia mínima por palabra=2, **learning rate = 0.03**, **épocas = 4**

```
loss 0.281, train acc 0.883, test acc 0.871
5268.7 examples/sec on [device(type='cuda', index=0)]
```



```
[125] predict_sentiment(net, vocab, 'this hotel is awesome')
```

```
'Positive'
```

```
[126] predict_sentiment(net, vocab, "I've been in better hotels, but it wasn't that bad either.")
```

```
'Negative'
```

```
[127] predict_sentiment(net, vocab, "the service leaves a lot to be desired")
```

```
'Negative'
```

[+ Código](#)[+ Texto](#)

```
[128] predict_sentiment(net, vocab, "the attention leaves much to be desired, especially the bathroom that was dirty")
```

```
'Negative'
```

```
[129] predict_sentiment(net, vocab, "the worst place i have been in my life")
```

```
'Negative'
```

```
[130] predict_sentiment(net, vocab, "I love this hotel")
```

```
'Positive'
```

```
[131] predict_sentiment(net, vocab, "I like this hotel")
```

```
'Negative'
```

```
[132] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it")
```

```
'Negative'
```

```
[133] predict_sentiment(net, vocab, "The hotel is too big, I couldn't fully explore it. I liked")
```

```
'Positive'
```

Se repite el problema de reconocer “I like this hotel”

CONCLUSIÓN

Lo más destacable es que LSTM da buenos resultados con tan solo 2.000 datos prediciendo frases más difíciles que GRU, por otro lado GRU tuvo problemas con “the service leaves a lot to be desired” (El servicio deja mucho que desear) en un entrenamiento con 4.000 reviews, sin embargo al tener más de 10.000 datos ambos modelos funcionan bien con las frases probadas.

Batch size resulta una variable bastante importante en el procesamiento de texto y en el tiempo de ejecución de los modelos. Un batch size demasiado alto con un gran número de datos puede hacer que el tiempo de ejecución sea más largo.

Learning Rate, la cantidad de épocas y la cantidad de datos se mantienen como fundamentales a la hora de conseguir un buen entrenamiento.

Finalmente se detectó que una baja cantidad de datos podría dar un buen entrenamiento, pero este entrenamiento puede no ser tan confiable como uno con muchos más datos.

REFERENCIAS

1. [16.1. Sentiment Analysis and the Dataset — Dive into Deep Learning 1.0.0-beta0 documentation \(d2l.ai\)](#)
2. [Datitos | Aprendizaje profundo 2022](#)