```
import torch
import reviewer


reviews = reviewer.load( "imdb_labelled.txt")
wordlist, worddict = reviewer.vocabulary( reviews)
seqs = reviewer.sequence( reviews, worddict)

P = len(reviews)      # Cada palabra tiene un indice...
N = len(wordlist)     # las sentencias son secuencias de enteros.
T = 5                 # El sentim. 1: pos, 0: neg.


class SRNN( torch.nn.Module):
    def __init__( _, isize, hsize, osize):
        super().__init__()
        _.context_size = hsize
        _.Wi = torch.nn.Linear( isize, hsize)      # Context es el...
        _.Wc = torch.nn.Linear( hsize, hsize)      # estado hidden...
        _.Wh = torch.nn.Linear( hsize, isize)      # anterior.
        _.Wo = torch.nn.Linear( hsize, osize)

    def forward( _, x0, h0):
        h1 = torch.tanh( _.Wi( x0) + _.Wc( h0))    # Recibe dos estimulos
        x1 = _.Wh( h1)
        return x1, h1        # x1: prediccion de la sig palabra.

    def predict( _, h):
        y = _.Wo( h)
        return y             # y: prediccion del sentimiento.

    def context( _, B=1):
        return torch.zeros( B, _.context_size)     # hidden inicial.


def one_hot( p, N):                 # Convierte el indice de la palabra...
    assert( 0 <= p < N)             # en un vector de ceros con solo un...
    pat = torch.zeros( 1, N)        # uno en esa posicion.
    pat[ 0, p] = 1.
    return pat
```

```
model = SRNN( N, 200, 2)
optim = torch.optim.SGD( model.parameters(), lr=0.01)
costf = torch.nn.CrossEntropyLoss()
                                    # Con CrossEntr podemos ver a cada...
                                    # palabra como una clase, y tambien...
model.train()                       # al sentim como una clase.
for t in range(T):
    E = 0.
    for b, (words, label) in enumerate(seqs[:750]):        # Incremental.
        error = 0.
        h = model.context()
        optim.zero_grad()
        for i in range( len( words[:-1])):         # Menos la ultima = '.'
            z = torch.tensor( words[i+1] ).view(1) # Tensor de dim=1
            x0 = one_hot( words[i], N)
            x1, h = model.forward( x0, h)          # h_t depende de h_t-1
            error += costf( x1, z)                 # Suma func de costo...
        y = model.predict( h)                      # para pred palabras...
        error += costf( y, torch.tensor(label))    # y el sentimiento.
        error.backward()
        optim.step()
        E += error.item()
        if b%100 == 0:
            print( t, b, error.item())
    print( E)


model.eval()
with torch.no_grad():
    r, t = 0, 0
    for words, label in seqs[750:]:
        h = model.context()
        for word in words[:-1]:
            x0 = one_hot( word, N)
            x1, h = model( x0, h)
        y = model.predict( h)
        r += y.argmax(1).item()==label[0]
        t += 1
print( r/t)
```

```
import torch
import reviewer


reviews = reviewer.load( "imdb_labelled.txt")
wl, wd = reviewer.vocabulary( reviews)
seqs = reviewer.sequence( reviews, wd)

P = len(reviews)             # Los lotes son matrices con las...
N = len(wl)                  # secuencias como filas, alineadas a...
T = 20                       # la derecha, completando con ceros.
B = 50
L = 75                       # L: Max long de sentencia.


pads = torch.LongTensor( P, L)       # Las seqs tienen que ser los...
for i in range(P):                   # indices de las palabras, es...
    seq = seqs[i][0]                 # importante que sean tipo Int.
    M = len(seq)
    pads[i] = 0                      # Pone 0s en toda la fila.
    if M<L:
        pads[i,-M:] = torch.tensor(seq)     # Si sobra lugar, a la derecha
    else:
        pads[i] = torch.tensor(seq[:L])     # Si falta lugar, lo corta.

labels = torch.tensor([ label for words,label in reviews], # Y este tiene
                      dtype=torch.float)                   # que ser float


trn_inputs =   pads[:-N//4]  # Se arman los cjtos de entrenamiento...
trn_target = labels[:-N//4]  # y validacion para las entradas y valores...
                             # objetivo con las padded seqs de antes...
tst_inputs =   pads[-N//4:]  # separando un cuarto de los datos del...
tst_target = labels[-N//4:]  # final. Igual con los sentiments.


from torch.utils.data import TensorDataset, DataLoader
                                                 # Se importa asi...
trn_data = TensorDataset( trn_inputs, trn_target)  # para poder armar...
tst_data = TensorDataset( tst_inputs, tst_target)  # el propio Loader.

trn_load = DataLoader( trn_data, shuffle=True, batch_size=B)
tst_load = DataLoader( tst_data, shuffle=True, batch_size=B)
```

```
class Sentiment( torch.nn.Module):
    def __init__( _, vocab, embed, context, output=1):
        super().__init__()
        _.isize = vocab      # Los embeddings pasan de la dim muy alta...
        _.esize = embed      # del vocabulario a una menor en donde las...
        _.hsize = context    # palabras afines tienen valores similares.
        _.osize = output
        _.embedding = torch.nn.Embedding( _.isize, _.esize)
        _.recurrent = torch.nn.GRU( _.esize, _.hsize, batch_first=True)
        _.output = torch.nn.Linear( _.hsize, _.osize)

    def forward( _, xi):             # En las RNN, GRU y LSTM tambien...
        xe = _.embedding( xi)        # se pueden usar varias capas...
        xo, h = _.recurrent( xe)     # dropout para generalizar mejor...
        y = _.output( h).sigmoid()   # y bidirec para context doble.
        return y.squeeze()


model = Sentiment( N, 100, 200, 1)
optim = torch.optim.Adam( model.parameters())
costf = torch.nn.MSELoss()


model.train()
for t in range(T):
    E = 0.
    for words, label in trn_load:
        optim.zero_grad()
        senti = model( words)               # No hace falta iterar a mano.
        error = costf( senti, label)        # El senti es una sigmoid.
        error.backward()
        optim.step()
        E += error.item()
    print( t, E)


model.eval()
with torch.no_grad():
    r, t = 0, 0
    for words, label in tst_load:    # Si senti>0.5 lo considero positivo.
        senti = model(words)
        r += ( (senti>=0.5)==(label==1) ).sum().item()
        t += len(label)
print( "Accuracy:", 100*r/t)
```