how we used to solve NLP tasks:

~2013

(sentiment analysis)

softmax layer
(predict positive)

this movie is great

1. randomly initialize the model parameters
   all trained from scratch

2. update all parameters by backprop using cross entropy loss from labeled training set

model has to learn how language works from only a small labeled dataset

~2014-2017

why train everything from scratch?
how can we leverage lots of unlabeled data?

softmax layer
(predict positive)
we can pretrain the $c_i$'s (word embs) using another objective fn that takes adv. of unlabeled data (self-supervised)

- Word2Vec, GloVe
- instead of starting w/ a random word embedding space, we start from a pretrained space in which word embs. capture some linguistic prop.
- train all other params from scratch $(W_h, W_c, W_o)$

~2018
- issues w/ word embeddings are static, only one vector per word type regardless of context
- the rest of the model (in our case, the RNN) is responsible for learning composition from scratch given just labeled data
what if we use the hidden states of a NLM instead of static word embeddings

softmax layer (predict positive)

contextualized word embs, still learn $W_h, W_e, W_o$ from scratch; "ELMo"

pretraining step! all params randomly Init trained w/self-sup objective

~2019 can we share more params than just the word embeddings?

- what about $W_h, W_e$
initialize my sentiment RNN w/ the LMs $W_h, W_e, c_1, \ldots, c_N$
  
  -- all we have to do is learn $W_0$ from scratch, all the other params are transferred

this is current paradigm in NLP, popularized by BERT model

-- we init our sentiment model w/ a pretrained NLM, and then
backprop the error from a drift task (downstream task) into these parameters. This is called fine-tuning.