

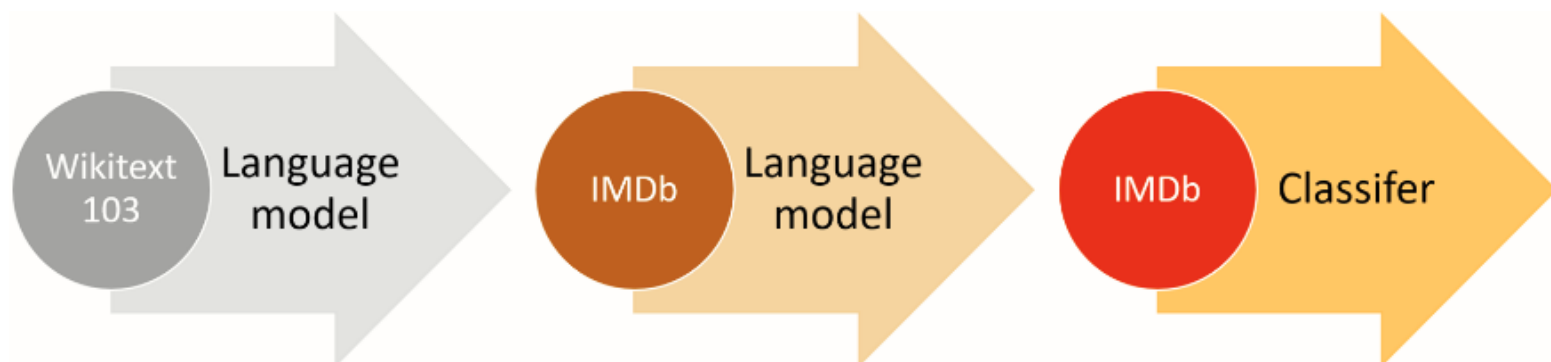
```
!pip install fastai
!pip install fastbook
```

```
from fastai import *
from fastbook import *
from fastai.text.all import *
```

▼ Movie Reviews Model

https://github.com/fastai/fastbook/blob/master/10_nlp.ipynb

- ULMFit Approach : **Extra Stage of fine_tuning before transfer learning**
 - This is known as the Universal Language Model Fine-tuning (ULMFit) approach. The paper showed that this extra stage of fine-tuning of the language model, prior to transfer learning to a classification task, resulted in significantly better predictions. Using this approach, we have three stages for transfer learning in NLP, as summarized in



▼ Detailed Notes on my Register

Each of the steps necessary to create a language model has jargon associated with it from the world of natural language processing, and fastai and PyTorch classes available to help. The steps are:

1. **Tokenization:** Convert the text into a list of words (or characters, or substrings, depending on the granularity of your model)
2. **Numericalization:** Make a list of all of the unique words that appear (the vocab), and convert each word into a number, by looking up its index in the vocab
3. **Language model data loader creation:** fastai provides an LMDataloader class which automatically handles creating a dependent variable that is offset from the independent variable by one token. It also handles some important details, such as how to shuffle the training data in such a way that the dependent and independent variables maintain their structure as required
4. **Language model creation:** We need a special kind of model that does something we haven't seen before: handles input lists which could be arbitrarily big or small. There are a number of ways to do this; in this

chapter we will be using a recurrent neural network (RNN). We will get to the details of these RNNs in the <>, but for now, you can think of it as just another deep neural network.

Tokenization :

Translating the original English language sequence into a simplified tokenized language—a language that is designed to be easy for a model to learn.

- Approach : Word Based

```
path=untar_data(URLs.IMDB)
```

100.00% [144441344/144440600 00:03<00:00]

```
#Path.BASE_PATH=path
```

```
#path.ls()
```

```
# get_text_files -> get all txt files from the path
# We can also optionally pass folders to restrict the search to a particular list of subfolders
files=get_text_files(path,folders=['test','unsup','train'])
```

```
txt=files[0].open().read()
txt
```

```
'Part zombie, part cannibal, part gore equals full of crap. Come on...it just doesn't work. Comma
ndo Norman Hopper(John Saxon)rescues a couple of Vietnam POWs who contracted a rare disease that
compels them to consume human flesh. One of the soldiers, Charlie Bukowski(John Morghen)bites Hop
per in the rescue. War hero Hopper will be plagued with cannibal instincts fighting their way to
the surface. Bukowski escapes from a veteran's psychiatric hospital and immediately goes into rel
ease and a gory rampage begins. A gun battle with police will comprise the meat (no pun intended)
```

```
len(txt)
```

806

et's try it out. We'll use fastai's coll_repr(collection, n) function to display the results. This displays the first n items of collection, along with the full size—it's what L uses by default. Note that fastai's tokenizers take a collection of documents to tokenize, so we have to wrap txt in a list:

```
spacy=WordTokenizer() # operating on space
toks=first(spacy([txt]))
print(coll_repr(toks,10))
```

```
(#147) ['Part','zombie',',',',','part','cannibal',',',',','part','gore','equals','full'...]
```

spacy -> separates a punctuation when in a word but not when in a number as :

```
first(spacy(['This is Rs1.00.']))
```

```
(#4) ['This', 'is', 'Rs1.00', '.']
```

fastai then adds some additional functionality to the tokenization process with the Tokenizer class:

- xx added with some words
 - For example, the first item in the list, `xxbos`, is a special token that indicates the start of a new text ("BOS" is a standard NLP acronym that means "beginning of stream"). By recognizing this start token, the model will be able to learn it needs to "forget" what was said previously and focus on upcoming words.

```
tkn = Tokenizer(spacy)
print(coll_repr(tkn(txt), 31))
```

```
(#172) ['xxbos', 'xxmaj', 'part', 'zombie', ',', ',', 'part', 'cannibal', ',', ',', 'part', 'gore', 'equals', 'full', '(',
```

List of Tokens

Here are some of the main special tokens you'll see:

1. `xxbos`:: Indicates the beginning of a text (here, a review)
2. `xxmaj`:: Indicates the next word begins with a capital (since we lowercased everything)
3. `xxunk`:: Indicates the word is unknown

Production Rules

```
defaults.text_proc_rules
```

```
[<function fastai.text.core.fix_html>,
 <function fastai.text.core.replace_rep>,
 <function fastai.text.core.replace_wrep>,
 <function fastai.text.core.spec_add_spaces>,
 <function fastai.text.core.rm_useless_spaces>,
 <function fastai.text.core.replace_all_caps>,
 <function fastai.text.core.replace_maj>,
 <function fastai.text.core.lowercase>]
```

1. `fix_html`:: Replaces special HTML characters with a readable version (IMDb reviews have quite a few of these)
2. `replace_rep`:: Replaces any character repeated three times or more with a special token for repetition

(xxrep), the number of times it's repeated, then the character

3. replace_wrep:: Replaces any word repeated three times or more with a special token for word repetition (xxwrep), the number of times it's repeated, then the word
4. spec_add_spaces:: Adds spaces around / and # rm_useless_spaces:: Removes all repetitions of the space character
5. replace_all_caps:: Lowercases a word written in all caps and adds a special token for all caps (xxup) in front of it
6. replace_maj:: Lowercases a capitalized word and adds a special token for capitalized (xxmaj) in front of it
7. lowercase:: Lowercases all text and adds a special token at the beginning (xxbos) and/or the end (xxeos)

```
coll_repr(tkn('&copy; Fast.ai www.fast.ai/INDEX'), 31)
```

```
'(#11) ['xxbos', '@', 'xxmaj', 'fast.ai', 'xxrep', '3', 'w', '.fast.ai', '/', 'xxup', 'index']'
```

Subword Tokenization

↳ 2 cells hidden

Numericalization

Numericalization is the process of mapping tokens to integers

1. Make a list of all possible levels of that categorical variable (the vocab).
2. Replace each level with its index in the vocab.

```
txts = L(o.open().read() for o in files[:2000])
txts
```

1. **txts -> 2000 words**
2. **txt -> first txt file**
3. **toks() create tokens**

```
txt
```

```
'Part zombie, part cannibal, part gore equals full of crap. Come on...it just doesn't work. Comma
ndo Norman Hopper(John Saxon)rescues a couple of Vietnam POWs who contracted a rare disease that
compels them to consume human flesh. One of the soldiers, Charlie Bukowski(John Morghen)bites Hop
per in the rescue. War hero Hopper will be plagued with cannibal instincts fighting their way to
the surface. Bukowski escapes from a veteran's psychiatric hospital and immediately goes into rel
apse and a gory rampage begins. A gun battle with police will comprise the meat (no gun intended)
```

created token for txt

```
toks = tkn(txt)
print(coll_repr(toks, 10))
```

```
(#172) ['xxbos', 'xxmaj', 'part', 'zombie', ',', ',', 'part', 'cannibal', ',', ',', 'part', 'gore'...]
```

Just like with SubwordTokenizer, we need to call setup on Numericalize; this is how we create the vocab. That means we'll need our tokenized corpus first. Since tokenization takes a while, it's done in parallel by fastai; but for this manual walkthrough, we'll use a small subset:

Vocab Training

txts -> 2000 words list

- mapped WRT tokenizer

```
toks200=txts[:200].map(tkn)
toks200[0]
```

```
(#172) ['xxbos', 'xxmaj', 'part', 'zombie', ',', ',', 'part', 'cannibal', ',', ',', 'part', 'gore'...]
```

We can pass this to setup to create our vocab:

```
num=Numericalize()
num.setup(toks200)
coll_repr(num.vocab,20)
```

```
'(#1968) ['xxunk', 'xxpad', 'xxbos', 'xaeos', 'xxfld', 'xxrep', 'xxwrep', 'xxup', 'xxmaj', 'the', '.', ',', ',', '
and', 'at', 'of', 'to', 'is', 'it', 'in', 'i', '1'
```

Our special rules tokens appear first, and then every word appears once, in frequency order.

- The defaults to Numericalize are,
 - min_freq=3,
 - max_vocab=60000
- This results in fastai replacing all words other than the most common 60,000 with a special unknown word token, xxunk. This is useful to avoid having an overly large embedding matrix, since that can slow down training and use up too much memory, and can also mean that there isn't enough data to train useful representations for rare words. However, this last issue is better handled by setting min_freq; the default min_freq=3 means that any word appearing less than three times is replaced with xxunk.

fastai can also numericalize your dataset using a vocab that you provide, by passing a list of words as the vocab parameter.

fastai can also numericalize your dataset using a vocab that you provide, by passing a list of words as the vocab parameter

Once we've created our Numericalize object, we can use it as if it were a function:

```
nums=num(toks)[:20]
nums
```

```
TensorText([ 2, 8, 209, 1432, 11, 209, 1433, 11, 209, 552, 0, 490, 14, 553,
10, 8, 173, 36, 89, 17])
```

can check that they map back to the original text:

```
' '.join(num.vocab[o] for o in nums)
```

```
'xxbos xxmaj part zombie , part cannibal , part gore xxunk full of crap . xxmaj come on ... it'
```

Text -> Batches for Language Model

IDEA :At every epoch we shuffle our collection of documents and concatenate them into a stream of tokens. We then cut that stream into a batch of fixed-size consecutive mini-streams. Our model will then read the mini-streams in order, and thanks to an inner state, it will produce the same activation whatever sequence length we picked.

- Kind of Data Augmentation techniques used in images

One Liner : Divide this array more finely into subarrays of a fixed sequence length. It is important to maintain order within and across these subarrays, because we will use a model that maintains a state so that it remembers what it read previously when predicting what comes next.

First Step

- To transform the individual texts into a stream by concatenating them together. As with images, it's best to randomize the order of the inputs, so at the beginning of each epoch we will shuffle the entries to make a new stream (**we shuffle the order of the documents, not the order of the words inside them, or the texts would not make sense anymore!**).

Second Step

- To cut this stream into a certain number of batches (which is our batch size). For instance, if the stream has 50,000 tokens and we set a batch size of 10, this will give us 10 mini-streams of 5,000 tokens. What is important is that we preserve the order of the tokens (so from 1 to 5,000 for the first mini-stream, then from 5,001 to 10,000...), because we want the model to read continuous rows of text (as in the preceding

example).

- An `xxbos` token is added at the start of each during preprocessing, so that the model knows when it reads the stream when a new entry is beginning.

This is all done behind the scenes by the `fastai` library when we create an `LMDataLoader`.

1. We do this by first applying our `Numericalize` object to the tokenized texts:
2. and then passing that to `LMDataLoader`:

```
nums200=toks200.map(num)
```

```
dl=LMDataLoader(nums200)
```

```
x,y=first(dl)  
x.shape,y.shape
```

```
(torch.Size([64, 72]), torch.Size([64, 72]))
```

looking at the first row of the independent variable, which should be the start of the first text:

```
' '.join(num.vocab[o] for o in x[0][:20])
```

```
'xxbos xxmaj part zombie , part cannibal , part gore xxunk full of crap . xxmaj come on ... it'
```

The dependent variable is the same thing offset by one token:

```
' '.join(num.vocab[o] for o in y[0][:20])
```

```
'xxmaj part zombie , part cannibal , part gore xxunk full of crap . xxmaj come on ... it just'
```

Training a Text Classifier

Two steps to training a state-of-the-art text classifier using transfer learning:

1. first we need to fine-tune our language model pretrained on Wikipedia to the corpus of IMDb reviews, and
2. then we can use that model to train a classifier.

Language Model Using DataBlock

```
# import shutil  
# shutil.rmtree('/root/.fastai/data/imdb_tok')
```

```
path.ls()
```

```
(#7) [Path('/root/.fastai/data/imdb/unsup'),Path('/root/.fastai/data/imdb/test'),Path('/root/.fastai/data/imdb/tmp_lm'),Path('/root/.fastai/data/imdb/imdb.vocab'),Path('/root/.fastai/data/imdb/train'),Path('/root/.fastai/data/imdb/README'),Path('/root/.fastai/data/imdb/tmp_clas')]
```

```
get_imdb=partial(get_text_files,folders=['test','unsup','train'])
```

```
dls=DataBlock(
    blocks=TextBlock.from_folder(path,is_lm=True),
    get_items=get_imdb,
    splitter=RandomSplitter(0.1)
).dataloaders(path,path=path,bs=128,seq_len=80)
```

```
dls.show_batch(max_n=5,unique=True)
```


	text	text_
0	xxbos i liked the first movie . xxmaj it was funny i guess . xxmaj but the second one , come on , my dad and xxup i , movie lovers , almost lost our dinner watching it . xxmaj at one point within this movie , xxmaj clara xxmaj topper who is looking through the creepy mansion for her husband is sitting at a table with her maid eating cake . xxmaj my dad and i made a xxmaj	i liked the first movie . xxmaj it was funny i guess . xxmaj but the second one , come on , my dad and xxup i , movie lovers , almost lost our dinner watching it . xxmaj at one point within this movie , xxmaj clara xxmaj topper who is looking through the creepy mansion for her husband is sitting at a table with her maid eating cake . xxmaj my dad and i made a xxmaj mystery
1	xxbos i liked the first movie . xxmaj it was funny i guess . xxmaj but the second one , come on , my dad and xxup i , movie lovers , almost lost our dinner watching it . xxmaj at one point within this movie , xxmaj clara xxmaj topper who is looking through the creepy mansion for her husband is sitting at a table with her maid eating cake . xxmaj my dad and i made a xxmaj	i liked the first movie . xxmaj it was funny i guess . xxmaj but the second one , come on , my dad and xxup i , movie lovers , almost lost our dinner watching it . xxmaj at one point within this movie , xxmaj clara xxmaj topper who is looking through the creepy mansion for her husband is sitting at a table with her maid eating cake . xxmaj my dad and i made a xxmaj mystery
2	xxbos i liked the first movie . xxmaj it was funny i guess . xxmaj but the second one , come on , my dad and xxup i , movie lovers , almost lost our dinner watching it . xxmaj at one point within this movie , xxmaj clara xxmaj topper who is looking through the creepy mansion for her husband is sitting at a table with her maid eating cake . xxmaj my dad and i made a xxmaj	i liked the first movie . xxmaj it was funny i guess . xxmaj but the second one , come on , my dad and xxup i , movie lovers , almost lost our dinner watching it . xxmaj at one point within this movie , xxmaj clara xxmaj topper who is looking through the creepy mansion for her husband is sitting at a table with her maid eating cake . xxmaj my dad and i made a xxmaj mystery
	xxbos i liked the first movie . xxmaj it was funny i guess	i liked the first movie . xxmaj it was funny i guess . xxmaj

Fine-Tuning the Language Model

1. To convert the integer word indices into activations that we can use for our neural network, we will use embeddings, just like we did for collaborative filtering and tabular modeling.
2. Then we'll feed those embeddings into a recurrent neural network (RNN), using an architecture called AWD-LSTM

This is handled automatically inside `language_model_learner`:

```
learn = language_model_learner(
    dls, AWD_LSTM, drop_mult=0.3,
    metrics=[accuracy, Perplexity()]).to_fp16()
```

 100.00% [105070592/105067061 00:01<00:00]

The loss function used by default is cross-entropy loss, since we essentially have a classification problem (the different categories being the words in our vocab). The perplexity metric used here is often used in NLP for language models: it is the exponential of the loss (i.e., `torch.exp(cross_entropy)`). We also include the accuracy metric, to see how many times our model is right when trying to predict the next word, since cross-entropy (as we've seen) is both hard to interpret, and tells us more about the model's confidence than its accuracy.

It takes quite a while to train each epoch, so we'll be saving the intermediate model results during the training process. Since `fine_tune` doesn't do that for us, we'll use `fit_one_cycle`. Just like `vision_learner`, `language_model_learner` automatically calls `freeze` when using a pretrained model (which is the default), so this will only train the embeddings (the only part of the model that contains randomly initialized weights—i.e., embeddings for words that are in our IMDb vocab, but aren't in the pretrained model vocab):

```
learn.fit_one_cycle(1, 2e-2)
```

epoch	train_loss	valid_loss	accuracy	perplexity	time
-------	------------	------------	----------	------------	------

0	4.006756	3.900314	0.301040	49.417946	26:40
---	----------	----------	----------	-----------	-------

epoch	train_loss	valid_loss	accuracy	perplexity	time
-------	------------	------------	----------	------------	------

0	4.006756	3.900314	0.301040	49.417946	26:40
---	----------	----------	----------	-----------	-------

Saving The State of Your Model

```
learn.save('1epoch')
```

```
Path('/root/.fastai/data/imdb/models/1epoch.pth')
```

This will create a file in `learn.path/models/` named `1epoch.pth`. If you want to load your model in another machine after creating your `Learner` the same way, or resume training later, you can load the content of this file with:

```
learn = learn.load('1epoch')
```

Once the initial training has completed, we can continue fine-tuning the model after unfreezing:

```
learn.unfreeze()
learn.fit_one_cycle(10, 2e-3)
```

 30.00% [3/10 1:26:24<3:21:36]

epoch	train_loss	valid_loss	accuracy	perplexity	time
0	3.763402	3.759718	0.317268	42.936314	28:34
1	3.697218	3.709552	0.322616	40.835491	28:40
2	3.637232	3.657323	0.328751	38.757435	29:08

 16.81% [442/2629 04:32<22:29 3.5169]

KeyboardInterrupt Traceback (most recent call last)

<ipython-input-36-a21773809809> in <module>()

1 learn.unfreeze()

----> 2 learn.fit_one_cycle(10, 2e-3)

22 frames

/usr/local/lib/python3.7/dist-packages/torch/_tensor.py in __iter__(self)

727 'iterations executed (and might lead to errors or silently give

,

728 'incorrect results).', category=torch.jit.TracerWarning,

stacklevel=2)

--> 729 return iter(self.unbind(0))

730

731 def __hash__(self):

KeyboardInterrupt:

 30.00% [3/10 1:26:24<3:21:36]

epoch	train_loss	valid_loss	accuracy	perplexity	time
0	3.763402	3.759718	0.317268	42.936314	28:34
1	3.697218	3.709552	0.322616	40.835491	28:40
2	3.637232	3.657323	0.328751	38.757435	29:08

 16.81% [442/2629 04:32<22:29 3.5169]

Once this is done, we save all of our model except the final layer that converts activations to probabilities of picking each token in our vocabulary. The model not including the final layer is called the encoder. We can save it with `save_encoder`:

```
learn.save_encoder('finetuned')
```

Text Generation

Before we move on to fine-tuning the classifier

```
TEXT="I didn't liked this movie"
N_WORDS=40
N_SENTENCES=2
preds=[learn.predict(TEXT,N_WORDS,temperature=0.75) for _ in range(N_SENTENCES)]
```

```
print("\n".join(preds))
```

```
i did n't liked this movie . I 've seen worse . It 's just that i was not entertained . It 's mos'
i did n't liked this movie . i have never seen a Korean movie before . This movie is just boring
```

Creating the Classifier DataLoaders

We're now moving from language model fine-tuning to classifier fine-tuning.

```
dls=DataBlock(
    #using voacb from our upper pretrained model
    #other its of now use if we create new vocab here
    blocks=(TextBlock.from_folder(path,vocab=dls.vocab),CategoryBlock),
    get_y=parent_label,
    get_items=partial(get_text_files,folders=['train','test']),
    splitter=GrandparentSplitter(valid_name='test')
).dataloaders(path,path=path,bs=128,seq_len=72)
```

The reason that we pass the vocab of the language model is to make sure we use the same correspondence of token to index. Otherwise the embeddings we learned in our fine-tuned language model won't make any sense to this model, and the fine-tuning step won't be of any use.

```
dls.show_batch(max_n=3)
```

	text	category
--	------	----------

0	xxbos xxmaj match 1 : xxmaj tag xxmaj team xxmaj table xxmaj match xxmaj bubba xxmaj ray and xxmaj spike xxmaj dudley vs xxmaj eddie xxmaj guerrero and xxmaj chris xxmaj benoit xxmaj bubba xxmaj ray and xxmaj spike xxmaj dudley started things off with a xxmaj tag xxmaj team xxmaj table xxmaj match against xxmaj eddie xxmaj guerrero and xxmaj chris xxmaj benoit . xxmaj according to the rules of the match , both opponents have to go through tables in order to get the win . xxmaj benoit and xxmaj guerrero heated up early on by taking turns hammering first xxmaj spike and then xxmaj bubba xxmaj ray . a xxmaj german xxunk by xxmaj benoit to xxmaj bubba took the wind out of the xxmaj dudley brother . xxmaj spike tried to help his brother , but the referee restrained him while xxmaj benoit and xxmaj guerrero	pos
---	--	-----

1 xxbos xxmaj by now you 've probably heard a bit about the new xxmaj disney dub of xxmaj miyazaki 's classic film , xxmaj laputa : xxmaj castle xxmaj in xxmaj the xxmaj sky . xxmaj during late summer of 1998 , xxmaj disney released " kiki 's xxmaj delivery xxmaj service " on video which included a preview of the xxmaj laputa dub saying it was due out in " 1 xxrep 3 9 " . xxmaj it 's obviously way past that year now , but the dub has been finally completed . xxmaj and it 's not " laputa : xxmaj castle xxmaj in

pos

Problem : Collating multiple documents into a mini-batch

create a mini-batch containing the first 10 documents. First we'll numericalize them:

```
nums_sample=toks200[:10].map(num)
```

```
nums_sample.map(len)
```

```
(#10) [172,227,520,315,262,181,94,217,77,442]
```

Issues :

Remember, PyTorch DataLoaders need to collate all the items in a batch into a single tensor, and a single tensor has a fixed shape (i.e., it has some particular length on every axis, and all items must be consistent).

- we used padding,resizing for images to solve this

Solution

1. **Padding** : We will expand the shortest texts to make them all the same size. To do this, we use a special padding token that will be ignored by our model. Additionally, to avoid memory issues and improve performance, we will batch together texts that are roughly the same lengths (with some shuffling for the training set). We do this by (approximately, for the training set) sorting the documents by length prior to each epoch. The result of this is that the documents collated into a single batch will tend to be of similar lengths. We won't pad every batch to the same size, but will instead use the size of the largest document in each batch as the target size.

The sorting and padding are automatically done by the data block API for us **when using a TextBlock, with is_lm=False**. (We don't have this same issue for language model data, since we concatenate all the documents together first, and then split them into equally sized sections.)

We can now create a model to classify our texts:

```
learn=text_classifier_learner(dls,AWD_LSTM,drop_mult=0.5,metrics=accuracy).to_fp16()
```

```
learn=learn.load_encoder('finetuned')
```

Fine Tuning the Classifier - finally ahhh

```
learn.fit_one_cycle(1, 2e-2)
```

epoch	train_loss	valid_loss	accuracy	time
-------	------------	------------	----------	------

0	0.359854	0.261566	0.896880	01:10
---	----------	----------	----------	-------

epoch	train_loss	valid_loss	accuracy	time
-------	------------	------------	----------	------

0	0.359854	0.261566	0.896880	01:10
---	----------	----------	----------	-------

In just one epoch we get the same result as our training

- We can pass -2 to freeze_to to freeze all except the last two parameter groups:

What is freezing in deep learning? Freezing a layer prevents its weights from being modified. This technique is often used in transfer learning, where the base model (trained on some other dataset) is frozen.

```
learn.freeze_to(-2)
```

```
#slice -> relative learning rates adaptive to layers of architecture
```

```
learn.fit_one_cycle(1, slice(1e-2/(2.6**4), 1e-2))
```

epoch	train_loss	valid_loss	accuracy	time
-------	------------	------------	----------	------

0	0.239247	0.181838	0.931600	01:18
---	----------	----------	----------	-------

epoch	train_loss	valid_loss	accuracy	time
-------	------------	------------	----------	------

0	0.239247	0.181838	0.931600	01:18
---	----------	----------	----------	-------

then we can unfreeze a bit more, and continue training:

```
learn.freeze_to(-3)
```

```
learn.fit_one_cycle(1, slice(5e-3/(2.6**4), 5e-3))
```

epoch	train_loss	valid_loss	accuracy	time
-------	------------	------------	----------	------

0	0.190908	0.161353	0.939280	01:30
---	----------	----------	----------	-------

epoch	train_loss	valid_loss	accuracy	time
-------	------------	------------	----------	------

0	0.190908	0.161353	0.939280	01:30
---	----------	----------	----------	-------

My Accuracy is 93.9% beacuse my total number of epochs were 3 - not 10 - to save my time

FASTAI paper - UMLFit

We reached 94.3% accuracy, which was state-of-the-art performance just three years ago. **By training another model on all the texts read backwards and averaging the predictions of those two models, we can even get to 95.1% accuracy**, which was the state of the art introduced by the ULMFiT paper. It was only beaten a few months ago, by fine-tuning a much bigger model and using expensive data augmentation techniques (translating sentences in another language and back, using another model for translation).

Using a pretrained model let us build a fine-tuned language model that was pretty powerful, to either generate fake reviews or help classify them. This is exciting stuff, but it's good to remember that this technology can also be used for malign purposes.

i can try another approach via training backwards and then fine tuning this model -