Lesson 4 Questions

1. What is "self-supervised learning"?

When you don’t give labels to a model, just feed it lots of data with a mechanism to make it’s own labels.

Usually self-supervised tasks can harder than classification. For example, feeding a model all the words in Wikipedia, and making it automatically label the next word in a sentence as the target, to try and predict it.

Self-supervised learning is usually done not to solve a specific task, but to use as a transfer model for a model that does aim to solve a specific task. For instance, you self-supervise train a model for the Wikipedia task, then use it as a classifier in another language-based task, say classifying reviews in IMDB as positive or negative.

But we can do even a step further. In the previous example, not all IMBD reviews are labelled as positive or negative. Many are just unlabelled, and so we can’t use it in our classifier, which is leaving good unused data.

Instead, we can self-supervise on Wikipedia to find the next word, then transfer the model to self-supervise on IMBD to also find the next word, and then finally transfer that model to be used as a classifier. This can result in significantly better performance. This approach is called Universal Language Model Fine-tuning (ULMFit).

1. What is a "language model"?

A language model is a model that has been trained to guess what the next word in a text is based on the previous words.

The Wikipedia model is a language model.

A language model is not a general term for just a model that is used in language, it’s specifically one that guesses the next word.

1. Why is a language model considered self-supervised?

Because we don’t give labels to the predicted next word, we make the model make the labels itself.

1. What are self-supervised models usually used for?

For be used as a transfer model in another model that is about language.

1. Why do we fine-tune language models?

Because the model that used the language model as a basis will likely have specific vocabulary and writing style that the language model has not been specifically trained for.

For example, in IMBD, there are names of actors, of movies etc, not used often (so not as important) in Wikipedia, and the way reviews are written are different to the way Wikipedia articles are written.

1. What are the three steps to create a state-of-the-art text classifier?

Find/train a general language model, transfer/fine tune it to make a task specific language model, transfer/fine tune it to make the target model for the task at hand.

E.g., we can self-supervise on Wikipedia to find the next word, then transfer the model to self-supervise on IMBD to also find the next word, and then finally transfer that model to be used as a classifier for positive or negative sentiment. This can result in significantly better performance. This approach is called Universal Language Model Fine-tuning (ULMFit).

1. How do the 50,000 unlabeled movie reviews help us create a better text classifier for the IMDb dataset?

Described in previous question.

1. What are the three steps to prepare your data for a language model?

(For a text, for example for a single IMDB review)

Tokenisation: Convert the text into a list of words/characters/substrings. The way you do this has to the same throughout all the three models. If you grab a model online, you have to tokenise your models in the same way.

Numericalization: First make a vocab list/dictionary of all the words used in the dataset. They will all be given unique numbers to identify. Then convert the tokenised text into a list of these numbers.

After we have converted all of our training and validation data into lists of numbers, we can use LMDataLoader to automatically label the last word(s)/token(s).

Not sure: see xxbos question. This creates tuples of the text without last word/token (the independent variable) and the last word/token (dependant variable).

As an aside, before for classification our data was always the same size. The input data was always a 28\*28 image. Now for language models we need our model to handle text, which can be of varying sizes. RNNs (recurrent neural networks) can do so.

1. What is "tokenization"? Why do we need it?

Tokenisation: Convert the text into a list of words/characters/substrings. The way you do this has to the same throughout all the three models. If you grab a model online, you have to tokenise your models in the same way.

But there are many details as to how we can tokenise. What about punctuation, is “don’t”, one token or two? And different languages which have different punctuation and rules.

We need tokenization as the model cannot understand text fed directly? It’s not formatted in a way any computer software can understand. For example, you can’t just give an image to a computer, you give it pixel information etc of an image.

1. Name three different approaches to tokenization.

Word-based: Split a sentence based on where the spaces are. Also: apply language specific rules to try and separate parts of meaning even if there aren’t spaces from ‘. Lastly: punctuation marks are generally split into separate tokens. In the sentence “don’t do that!” I think it would be “do””n’t””do””that””!”.

Why do we split “don’t” into “do””n’t”? It’s intuitive, e.g.: “it’s” is short for “it is”, which is really just two words, so makes more sense to be two tokens.

Subword-based: split words into smaller parts based on the most commonly occurring substrings. E.g. “occasion” into “o””c””ca””sion”. This works for all languages, and even ‘languages’ such as genome sequences and music notation. Maybe there’s a use here for sound too?

Character-based: simply just split the sentence into its individual characters.

1. What is xxbos?

It is an example of a special token. It indicates the start of the first item in a text, so that the model knows it should ignore the previous text and predict only from then onwards. Or that it should focus on the text from then onwards more for the prediction?

I’m confused whether this A. is how the model knows how the data is structure I.E. where each text starts and ends, so doesn’t use the previous text at all for the prediction, or B. is to tell the model that the previous text isn’t as important, but might be. E.g. If in a full review, the xx indicates a new sentence is being started, so focus on the new sentence’s words.

Actually it might be A. since it’s only for new text, not new sentences. LMDataLoader might store text as tuples, I’m not sure.

1. List four rules that fastai applies to text during tokenization.

We want the tokenized text to be designed in a way so that the model can learn from it easily and efficiently. Fastai has some rules, or special tokens, to do so:

xxbos indicates the start of a new text (e.g. a IMBD review) (NOT A NEW SENTENCE IN A TEXT). This might be how LMDataLoader knows when text starts and ends to know not to use previous text in the current prediction.

xxmaj indicates the next word begins with a capital. We do this so that our vocab list doesn’t store separate entries for words with capitals at the start. E.g. Start and start. This saves memory, but also might be good as both uppercase and lowercase variants have similar meaning, but the model knows the former is capital so can learn accordingly.

Xxunk indicates the word is unknown.

Xxrep and xxwrep are for repeated characters and words, as when repeated characters/words could have a different meaning than the same word just multiple times, e.g., “A big big barn”, for emphasis. Or “A big barn!!”. This also can save space in memory when storing tokenised lists.

There are also more rules.

1. Why are repeated characters replaced with a token showing the number of repetitions and the character that's repeated?

Xxrep and xxwrep are for repeated characters and words, as when repeated characters/words could have a different meaning than the same word just multiple times, e.g., “A big big barn”, for emphasis. Or “A big barn!!”. This also can save space in memory when storing tokenised lists.

1. What is "numericalization"?

Numericalization: First make a vocab list/dictionary of all the words used in the dataset. They will all be given unique numbers to identify. Then convert the tokenised text into a list of these numbers. More formally: mapping tokens to integers.

1. Why might there be words that are replaced with the "unknown word" token?

Because the transfer model might not have those words in its vocab list. They could be slang for example.

Or because there is a limit set to know many words the vocab list can store. Say only the most common 60000 words. Then even previously encountered words that were not frequent enough would show up as xxunk or “unknown word”. Alternatively, the vocab list could have only stored words that appear >n times.

Extra Q: how is the whole input stream separated into batches and shuffled?

I think:

Say for IMDB reviews. Firstly, we get all the reviews, shuffle them into a random order, and concatenate them all together into one big stream of tokens.

Say the stream has 50,000 tokens and a batch size of 10. This gives us 10 mini-streams of 5,000.

We preserve the mini-stream token allocation, I.E, mini-stream 1 always has tokens 1-5000 etc. The model knows the start of a mini-stream from a xxbos token indicating it.

The model then trains on the mini-streams in order, from 1 to 10.

“and thanks to an inner state, it will produce the same activation whatever sequence length we picked.?”

1. With a batch size of 64, the first row of the tensor representing the first batch contains the first 64 tokens for the dataset. What does the second row of that tensor contain? What does the first row of the second batch contain? (Careful—students often get this one wrong! Be sure to check your answer on the book's website.)

My answer is wrong. I need to spend more time working on tensors. The official answer is:

a. The dataset is split into 64 mini-streams (batch size)

b. Each batch has 64 rows (batch size) and 64 columns (sequence length)

c. The first row of the first batch contains the beginning of the first mini-stream (tokens 1-64)

d. The second row of the first batch contains the beginning of the second mini-stream

e. The first row of the second batch contains the second chunk of the first mini-stream (tokens 65-128)

My answer:

A batch size of 64 means there will be 64 sequences of tokens in a batch.

So, if the first row of the first batch was the first 64 tokens, the second row of the first batch would simply be the second 64 tokens. In other words, the 65-128 tokens.

Just like:

Graphical user interface, application

Description automatically generated

And the first row of the second batch?

Well if the first batch is complete, so 64\*64 tokens, 4096, have been processed already. So the first row of the second batch should be the tokens 4097-4160.

1. Why do we need padding for text classification? Why don't we need it for language modeling?

In classification, text or image, PyTorch DataLoaders needs to collect all the items in a batch to a single tensor.

I think this means, because a tensor can only store items that are of the same type and size, we need padding to make all the documents (e.g. IMDB reviews) the same size. E.g. a tensor can store images of 28\*28 pixels in size, but it cannot deal with images of varying sizes, say one is 28\*28 and another is 256\*256.

“and thanks to an inner state, it will produce the same activation whatever sequence length we picked.?”

We put documents of similar sizes together in batches, and pad them all to be the size of the largest document in any given batch.

The reason why we pad, is because naturally cropping documents will lose important information.

Put simply, for language modelling, all the documents are concatenated so there are no variations in the sizes of items in batches.

In language modelling, we use Fastai LMDataLoader, which shuffles all the documents and concatenates them into one large stream/string of tokens. This stream is then cut into n fixed-size mini-streams. N is the batch size. E.g. 50,000 token stream into 10 mini streams of 5000, the batch size is standardised at 10 tokens per batch. We retain mini stream order. So the model starts on the first batch, the first 10 tokens of mini stream 1, then the first batch, the second 10 tokens of mini stream 1 etc until it’s done with mini stream 1 and starts the first batch of 10 tokens in mini stream 2.

There is no need for padding. Say the batch size is 6. Each item is a list of tokens of the same size. Tensors can store items that are the same type and size fine? E.g:

Graphical user interface, application

Description automatically generated

This is a batch size of 6.

1. What does an embedding matrix for NLP contain? What is its shape?

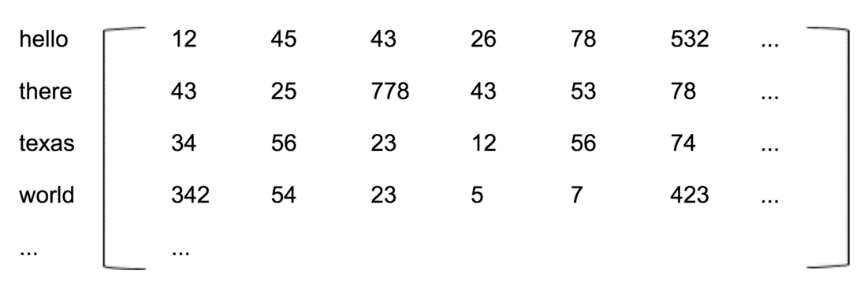
Embedding matrixes are a little complicated.

The official answer is:

It contains vector representations of all tokens in the vocabulary. The embedding matrix has the size (vocab\_size x embedding\_size), where vocab\_size is the length of the vocabulary, and embedding\_size is an arbitrary number defining the number of latent factors of the tokens.

My answer is:

An embedding matrix is related to the vocab in an NLP. It is a list of all the words and their corresponding embeddings. An embedding is a number, and it is strange.



For the first row, the word “hello” has it’s embedding listed. Each embedding ‘identifies’ a word in some way. They represent some kind of relationship between that word and others. Each row is very long, say 300 long, making the matrix 300 dimensions. To clarify, the embedding is the list of all the numbers on a row.

More info at https://petuum.medium.com/embeddings-a-matrix-of-meaning-4de877c9aa27

How to create an embedding matrix:

Say we have a pretrained language model. The embedding matrix will have a list of the language model’s vocab and their embeddings (their relationships to eachother). We encounter a new word ‘hello’. As it’s not already in the model, we simply just initialise a row for it with a random embedding (a random vector), aka setting its relationships to other words at random.

We then use the embedding matrix as the first layer for a neural network for NLP.

1. What is "perplexity"?

It’s a performance metric like accuracy often used to judge NLP models.

It is the exponential of the loss. In our case, the loss function by default is cross\_entropy, so it’s e^cross\_entropy.

1. Why do we have to pass the vocabulary of the language model to the classifier data block?

Because the correspondence order between the tokens and their indexes must be preserved from the transfer model to the new one. What I call the transfer model is called the encoder. All we do is cut off the head of the encoder and train again for the target task. SaveEncoder, rather than SaveModel, cuts off the head.

This correspondence order is needed so the model can properly use the embedding matrix created during the encoder’s fine tuning.

1. What is "gradual unfreezing"?

Gradual unfreezing is when we ‘unfreeze’ parameters, aka we only let some learn and others not, and then slowly let more learn until unfreezing entirely.

It is unfreezing one layer at a time, then training that layer’s parameters, then unfreezing another layer and then training, etc.

For computer vision, usually we just unfreeze all the parameters at once, aka we let all the parameters learn straight away. In NLP transfer learn classifiers, it turns out that unfreezing some parameters, letting them learn a bit, then unfreezing more, then learning, then unfreezing all and learning, can make a real difference.

1. Why is text generation always likely to be ahead of automatic identification of machine-generated texts?

Because classifier identification models can be used to create better generation models. And kinda by definition, a better classifier model can only be trained on a better generation model after it has already been released online.