COMP3220 — Document Processing and the Semantic Web

Week 06 L1: Advanced Topics in Deep Learning

Diego Mollá

COMP3220 2021H1

Abstract

This is the final lecture on deep learning where we will introduce several advanced topics on deep learning for text processing. The emphasis here is on aspects related to the generation of text. We will see an approach that generates text by learning a language model based on a corpus, and we will advance some topics on the use of encoding and decoding architectures that are able to generate text based on some input context. This can be used in multiple tasks, such as machine translation (e.g. French to English, text summarisation (from text to a summary), or even caption generation (from an image to text). We will conclude with open challenges in deep learning that are the subject of current research.

Update March 28, 2021

Contents

Reading

- Deep Learning book, section 8.1.

Additional Reading

- Jurafsky & Martin, Chapter 9.
- The Illustrated BERT, ELMo, and co.: `http://jalammar.github.io/illustrated-bert/`

1 Text Generation

Generating Text Sequences

- One of the advances of deep learning versus shallower approaches to machine learning is its ability to process complex contexts.
- This has allowed significant advances in image and text processing.
- We have seen how to process text sequences for text classification.

Text generation as a particular case of text classification

- Given a piece of text ...
- Predict the next character.

Text Generation as Character Prediction

- Our training data is a set of samples of the form:

  Input Text fragment.

  Label Next character to predict.

- We do not need to manually annotate the training data: the data are self-annotated.
- This means that we can easily gather training data for text generation.
- This is the idea for training language models (next slide).

Language Models

- Given a collection of text, we can train a language model that can be used to generate text in the same style.

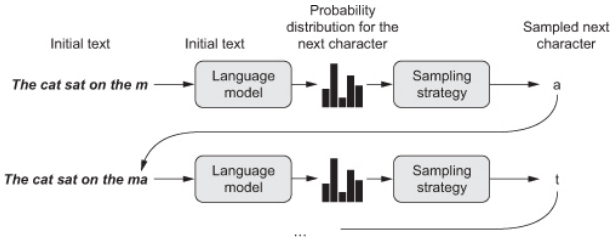


Figure 8.1 of Chollet (2018).

Implementing Character-level LSTM Text Generation

- The architecture of the model is of the kinds we have seen for text classification.
  - The input is a sequence of characters.
  - The "class" to predict is the next character to generate.
- If we add an embedding layer after the input, This layer will learn character embeddings.

```
model = tf.keras.models.Sequential()
model.add(layers.Embedding(len(chars), 20, input_len=maxlen))
model.add(layers.LSTM(128))
model.add(layers.Dense(len(chars), activation='softmax))
```

The architecture above shows an example of how you could learn character embeddings: you could train the network on a large collection of texts (for example, the entire Wikipedia). Then, you can save the weights of the Embedding layer. These weights can then be loaded in another network.

A very similar variant of the architecture can be used to learn word embeddings, by training the network to predict the next word (not the next character). The final layer would then need to have as many cells as the entire vocabulary. This creates some practical problems with scalability and sparse data which

need to be addressed when training the network. Solving these problems of scalability and sparse data is beyond the scope of this unit. If you want to know more about this, read, for example, this blog post: `https://jalammar.github.io/illustrated-word2vec/`

But you can always use the word embeddings made publicly available, as we have seen in a previous lecture.

Generating Text

- Remember that the output of a prediction is a probability distribution.
- To generate the next character, we can sample from the probability distribution.
- We can determine how deterministic the sampling is:
  - We can always return the character with highest probability ...
  - Or we can select a character randomly ...
  - Or we can do something in between, according to a "temperature" parameter.

```
import numpy as np
def reweight_distribution(original_distribution, temperature=0.5):
    distribution = np.log(original_distribution) / temperature
    distribution = np.exp(distribution)
    return distribution / np.sum(distribution)
```

Figure: Different Reweightings

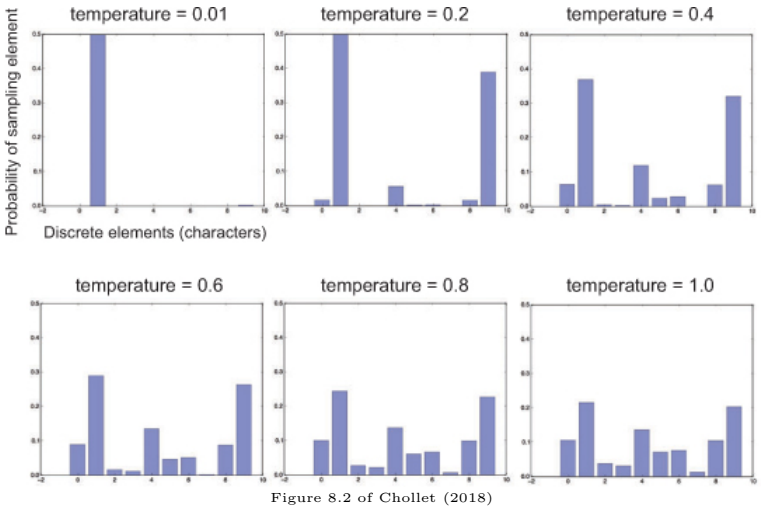


Figure 8.2 of Chollet (2018)

The figure shows the reweightings of a sample distribution as we change the temperature. Low temperature will generate a deterministic distribution where only one value has probability near 1, and the other values have probabilities near 0. In contrast, high temperature will generate probabilities that are nearly identical, simulating random choice.

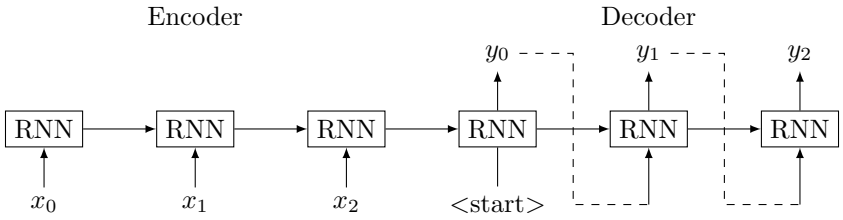
Example

See notebook ...

2 Encoder-Decoder Architecture

The Encoder-Decoder Architecture

- Composed of an encoder and a decoder.
  - The encoder can be an RNN chain that takes the input.
  - The decoder can be an RNN that takes the output of the previous RNN as input.
- Revolutionised machine translation and many other text processing applications.
- The encoder stage can be something non-textual, e.g. images for caption generation.



The encoder-decoder architecture is a general architecture that can be used for any case where the desired output is a sequence of length different from the input sequence. It can even be used to generate text based on non-textual information, such as image caption generation.

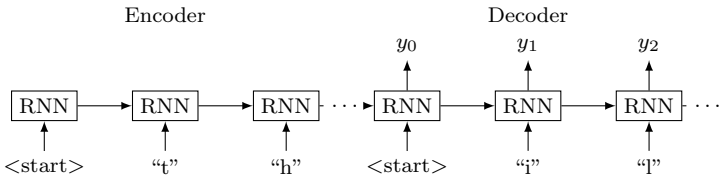
In the most basic approach, the encoder-decoder architecture can be implemented as two RNN layers: the encoder is an RNN layer that generates an output. This outut is then the input to the decoder. Many variants and enhancements of this architecture are being proposed.

Training the Encoder-Decoder Architecture

A common approach to train the encoder-decoder architecture is to apply teacher forcing:

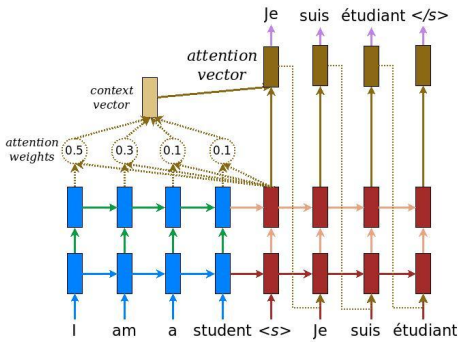
- Use the target sequence to guide the training of the decoder.
- For example, in an English to French machine translation system, we feed the target French translation to the decoder.

"The weather is fine" $\longrightarrow$ *"Il fait bon"*



Attention: An Improvement to the Encoder-Decoder Architecture

Attention is an enhancement in the seq2seq architecture that allows to focus on parts of the input during the generation stage by the decoder.

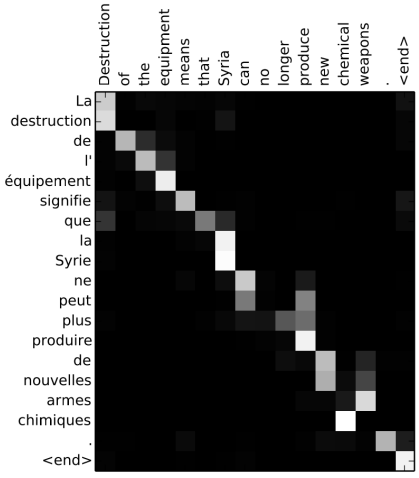


https://github.com/tensorflow/tensorflow/blob/r1.13/tensorflow/contrib/eager/python/examples/nmt_with_attention/nmt_with_attention.ipynb

The encoder and decoder of this image consist of two stacked RNN chains, and attention on the top RNN chain of the encoder is used to predict the next output of the decoder. We can also see that the encoder and decoders operate on words and not on characters, in contrast with the previous examples of this lecture. The image pictures attention when generating the first word. The same procedure would be used to generate the subsequent outputs.

Attention for MT

Very useful to start understanding the decision processes of the model.



Bahdanau et al. (2015) arXiv:1409.0473

Attention in Caption Generation

A woman is throwing a frisbee in a park.

Xu et al. (2015) arXiv:1502.03044

3 Pre-training and Fine-tuning

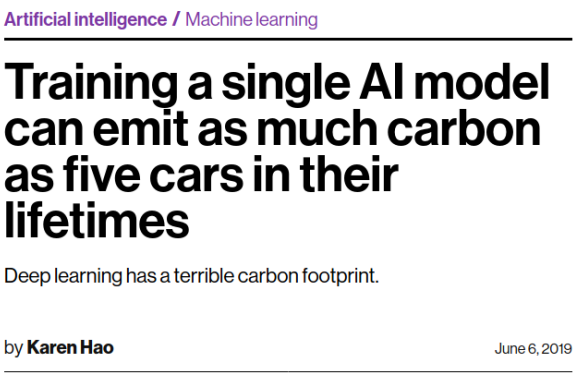
Problems with Supervised Learning

Annotated data

- Supervised learning requires (a lot of) annotated data.
- Annotated data can be costly.
- Human annotated data can contain annotation errors.

Training size

- Supervised learning requires a lot of (annotated) data.
- Large companies can afford the resources for processing large volumes of data, others can't.
- Some domains do not have much text anyway.



Artificial intelligence / Machine learning

Training a single AI model can emit as much carbon as five cars in their lifetimes

Deep learning has a terrible carbon footprint.

by **Karen Hao** June 6, 2019

`https://www.technologyreview.com/2019/06/06/239031/training-a-single-ai-model-can-emit-as-much-carbon-as-five-cars-in-their-lifetimes/`

Idea: Pre-Train and Fine-Tune

Pre-training

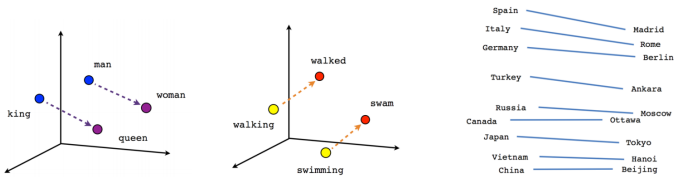
- Develop a system that can be trained with large volumes of data.
- Make the system as general as possible, so that it can be used for multiple tasks.

Fine-tuning

- Design a Deep Learning model that contains:
  - A layer pre-trained for a general task.
  - Additional layers that adapt the general task to our specific task.
- Fine-tune the system using the (smaller) training data of our specific task.

Example: Word Embeddings

- As we have seen in a previous lecture, word embeddings can be learnt using large, unlabelled data.
- These pre-trained word embeddings can be used to initialise an embeddings layer in our Deep Learning model.
- We have the choice to update these word embeddings, or not.



Male-Female Verb tense Country-Capital

Huggingface's transformers library

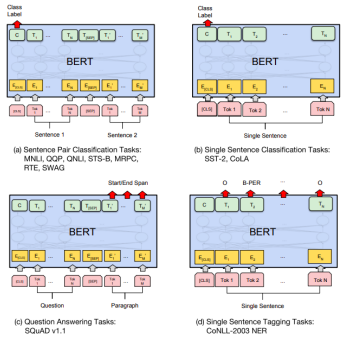
Transformers

`https://github.com/huggingface/transformers`

- Huggingface's transformers library contains a large repository of pre-trained models.
- These models are contributions from many researchers and developers.
- These models are being used to obtain state-of-the-art results.

Example: Using BERT in Keras

- BERT is one of the most popular architectures for pre-training and fine-tuning.
- Look at the lecture notebook for an example of use in keras.
- BERT is easy to use, but fine-tuning can take a long time.



`http://jalammar.github.io/illustrated-bert/`

Take-home Messages

1. Text generation as a task of character (or word) prediction.
2. Describe the encoder-decoder architecture. What is this architecture good for?
3. What is teacher forced training and what is it good for?
4. Transfer learning and fine-tuning.

What's Next

Weeks 7-12

- Semantic Web (Rolf Schwitter).
- Assignment 2 submission deadline on Friday 23 April 2021.