Shakespeare generation with the `romanesco` RNN language model (MT Übung 4)

Margaret Chi

Motiviation, data and preprocessing

The Infinite Monkey Theorem posits that a monkey working at a typewriter for an infinite amount of time will reproduce the works of William Shakespeare (Tenen, 2017). Without the luxuries of infinite time, a monkey or a typewriter, for this exercise I wanted to test the ability of the `romanesco` language model to construct Shakespeare-like text based on the plays.

The data source was an XML encoding of the PlayShakespeare editions (Severdia, 2011).

Preprocessing was performed to produce input appropriate for `romanesco`:

1. Extract content (lines that are actually spoken) from each `<speech>` element.
2. Concatenate lines of each speech into a block, encoding original line endings with a special token `<LBR>` (rationale: help `romanesco` learn Shakespearean meter).
3. Use `nltk.sent_tokenize` to tokenize each speech block into sentences.
4. Output one sentence per line to a text file.

The 37 preprocessed plays equated to approximately 4.6MB of text.

Training and `romanesco` changes

Tokenization/identifying vocabulary items

By default, `romanesco` tokenized text based on whitespace, analyzing sequences like *fight* and *fight!* as distinct vocabulary items. Based on the theory that tokenizing *fight!* as *fight* and *!* instead would improve data about contexts for the *fight* token, the `read_words` function was modified to identify tokens via pattern matching. The resulting regular expression for tokens accounts for usage of punctuation marks in the PlayShakespeare texts:

```
    TOKEN_PATTERN = '[,;:!?.“”()—]|[^,;:!?.“”()—\s]+'
```

This change reduced perplexity when training and scoring data totalling around 1.5MB.

Experimentation

Subsequent experiments with hyperparameters and data were motivated by these ideas:

- Larger datasets ⟹ more data points to learn from
- Larger model size (e.g. more hidden layer nodes) ⟹ greater learning capacity
- More training epochs ⟹ more opportunity to learn from training data
- Caveat: too many model parameters or excessive training cause overfitting, i.e. model becomes optimized for training data but gives poor results for other data

As expected, including more plays during training gave lower perplexity on development data. However, working with the `romanesco 0.1` defaults as a baseline, training became prohibitively slow, e.g. an hour for 3.1MB of data. The new default hyperparameters for `romanesco 0.2` (e.g. larger batch size) allowed training to continue with the full 4.6MB of play text.

Early stopping

To prevent overfitting during further experiments with `romanesco 0.2`, a basic early stopping algorithm was added to the `train` function:

1. After every `val_epochs` epochs, check model performance against a validation dataset. Save the model if it gives the best validation performance so far.
2. Terminate training after `patience` consecutive checks with no validation improvement.
3. Retain the `epochs` parameter as a hard stopping condition for training.

The initial implementation saved the latest model to a checkpoint file in `train`, then called the `score` function to load the model and score it for the validation data. Unfortunately, this approach led to a memory leak: it seems the `tf.Graph` created by each `score` call was not cleaned up, even after the `tf.Session` used to run it went out of scope. As a result, the final implementation instead duplicates some of the `score` code in the `train` function.

Comparison of initial and final model settings and performance

		Initial	Final	Comments
Dataset size	In plays	11	37	More data $\implies$ better model.
	In disk space	1.5MB	4.6MB	Final model data split into training, validation (used for early stopping), and test sets.
Hyperparams	Vocab size	10000	10000	Complete works contain ~28,000 types, but ~12,500 are singletons. Models did not benefit from vocab size > 10000.
	Embedding size	1500	256	
	Hidden size	1500	1024	Hidden size of 2048 led to overfitting.
	Batch size	20	32	Smaller value than `romanesco 0.2` default (64) gave performance benefit, with only small training time increase due to more parameter updates.
	Epochs	10	50	Final setting = upper limit for training. Early stopping ended training after 13 epochs.
	Time steps	35	100	25 gave better performance in early tests, but training became too slow.
Results	Perplexity	206.76	72.47	

References

Severdia, R. (2011). *PlayShakespeare.com-XML.* Retrieved from `https://github.com/severdia/PlayShakespeare.com-XML`

Tenen, D. (2017). Unintelligent design. *boundary 2*, *44*(2), 145-156.