Can we talk?

Neural Machine Translation with Keras.

Bojan Božić

Centre for Applied Data Analytics (CeADAR)

PyCon Ireland 2018

Introduction

- NMT has highly promising performance for large training data.
- Common principle: encoding meaning of input into concept space and performing translation based on encoding $\rightarrow$ deeper understanding and learning of translation rules, better translation than SMT.
- Problem: tendency towards overfitting to frequent observations and overlooking special cases.
- Cause: Translational function is shared, so high- and low-frequency pairs impact each other by adapting shared parameters. Smoothness of translation function makes infrequent pairs seem like noise.

Neural Machine Translation

Is an approach to machine translation that uses a large neural network. It departs from phrase-based statistical approaches that use separately engineered subcomponents. E.g. Google uses Google Neural Machine Translation (GNMT) in preference to its previous statistical methods.

Problem: Low-frequency pairs

src.	人类共有二十三对染色体。
ref.	Humans have 23 pairs of chromosomes.
NMT	There are 23-year history of human history.

Table 1: An example of Chinese-to-English meaning drift with NMT.

Errors


带好孩子
Take the child
小心落水
Fall into water carefully


中國東方航空 上海航空公司
CHINA EASTERN SHANGHAI AIRLINES
请在一米线外等候
Please wait outside rice-flour noodle
www.engrish.com


文明观赏
请勿伤蛇
www.engrish.com
Do not hurt the snake viewing civilization


公共旅游卫生间
Public Toilet Tourism
Alamy

Statistic Machine Translation

- Based on statistics of words and phrases (i.e. symbolic method with discrete model).
- Discrete model = probability of infrequent pairs cannot be smoothed out.
- Lack of shared parameters = frequent pairs have much less impact on infrequent pairs.
- SMT memorises as many observed patterns as possible by using a phrase table.
- Ideal: Neural model with complementary statistical support.

Attention-based NMT

- Baseline: Attention-based RNN model with encoder-decoder frame.
- MLP similarity function: $\alpha_{ij} = \frac{e_{ij}}{\sum e_{ik}}; e_{ij} = a(s_{i-1}, h_j)$
- Semantic content: $c_i = \sum \alpha_{ij} h_j$
- Update with recurrent function: $s_i = f_d(y_{i-1}, s_{i-1}, c_i)$
- Next word: $p(y_i) = \sigma(y_i^T W z_i)$
- Intermediate variable: $z_i = g(y_{i-1}, s_{i-1}, c_i)$

OOV Treatment

- Manually defined dictionary specifies how to translate OOV words.
- Use this to construct local memory at run-time.
- When OOV is encountered the vector of a similar word is borrowed.
- Alternative choices should prevent collisions of the similar word.
- Problem: Vocabulary of neural model is fixed $\rightarrow$ no probabilities.
- Solution: Rewrite similar word by OOV word and redirect prediction.

Translation Results

src.	人类共有二十三对染色体。
ref.	Humans have 23 pairs of chromosomes.
Moses	A total of 23 human chromosome.
NMT	There are 23-year history of human history.
M-NMT	There have a total of 23 species of chromosomes.

Table 2: The translations from different systems for the Chinese-to-English 'meaning drift' example.

Google

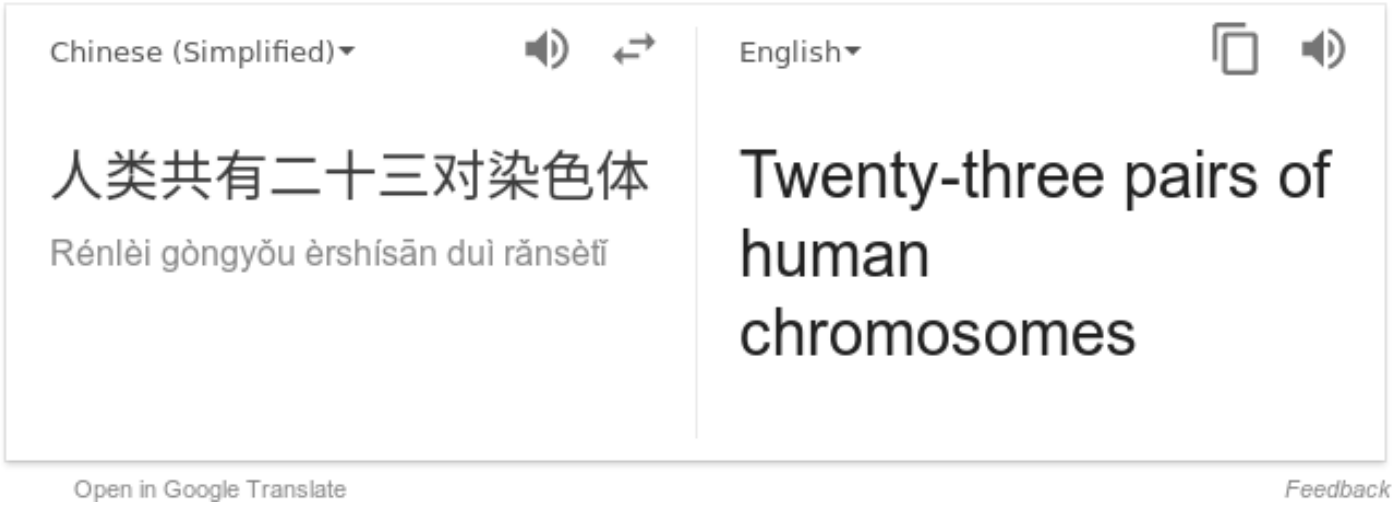


Figure 1: Even Google Translate does a decent job.

Keras

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Keras Principles

- **User friendliness.** Best practices for reducing cognitive load: it offers consistent and simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.
- **Modularity.** Neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.
- **Easy extensibility.** New modules are simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.
- **Works with Python.** No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

A Brief Intro to Keras (1)

The core data structure of Keras is a model, a way to organize layers. The simplest type of model is the `Sequential` model, a linear stack of layers. For more complex architectures, see Keras functional API, which allows to build arbitrary graphs of layers.

Here is the Sequential model:

```
from keras.models import Sequential

model = Sequential()
```

A Brief Intro to Keras (2)

Stacking layers is as easy as `.add()`:

```
from keras.layers import Dense

model.add(Dense(units=64, activation='relu',
                input_dim=100))
model.add(Dense(units=10, activation='softmax'))
```

Once your model looks good, configure its learning process with `.compile()`:

```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

A Brief Intro to Keras (3)

Further configuration of the optimiser is made reasonably simple, while allowing the user to be fully in control when she needs to (the ultimate control being the easy extensibility of the source code).

```
model.compile(loss=keras.losses.categorical_
                  crossentropy,
                  optimizer=keras.optimizers.
                  SGD(lr=0.01, momentum=0.9,
                  nesterov=True))
```

A Brief Intro to Keras (4)

You can now iterate on your training data in batches:

```
# x_train and y_train are Numpy arrays --just
# like in the Scikit-Learn API.
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

Alternatively, you can feed batches to your model manually:

```
model.train_on_batch(x_batch, y_batch)
```

A Brief Intro to Keras (5)

Evaluate your performance in one line:

```
loss_and_metrics = model.evaluate(x_test, y_test,
                                  batch_size=128)
```

Or generate predictions on new data:

```
classes = model.predict(x_test, batch_size=128)
```

A Neural Machine Translation System from Scratch

What we need or have to do:

1. Translation Dataset.
2. Prepare the text data.
3. Train Neural Translation model.
4. Evaluate Neural Translation model.

The Dataset

German-English dataset from `https://tatoeba.org/eng` with tab-delimited bilingual sentence pairs.

```
bojan@soc-LUJGDT6H2-B:~/git/machine-translation$ head deu.txt
Hi.	Hallo!
Hi.	Grüß Gott!
Run!	Lauf!
Fire!	Feuer!
Help!	Hilfe!
Help!	Zu Hülf!
Stop!	Stopp!
Wait!	Warte!
Go on.	Mach weiter.
Hello!	Hallo!
```

Figure 2: Translation dataset.

Preparing Text Data

What's the problem with raw data?

- There is punctuation.
- The text contains uppercase and lowercase.
- There are special characters in the German.
- There are duplicate phrases in English with different translations in German.
- The file is ordered by sentence length with very long sentences toward the end of the file.

Clean Text

```
bojan@soc-LUJGDT6H2-B:~/git/machine-translation$ python clean_text.py
Saved: english-german.pkl
[hi] => [hallo]
[hi] => [gru gott]
[run] => [lauf]
[fire] => [feuer]
[help] => [hilfe]
[help] => [zu hulf]
[stop] => [stopp]
[wait] => [warte]
[go on] => [mach weiter]
[hello] => [hallo]
[i ran] => [ich rannte]
[i see] => [ich verstehe]
[i see] => [aha]
[i try] => [ich probiere es]
[i won] => [ich hab gewonnen]
[i won] => [ich habe gewonnen]
[smile] => [lacheln]
[cheers] => [zum wohl]
[freeze] => [keine bewegung]
[freeze] => [stehenbleiben]
[got it] => [verstanden]
```

Figure 3: Removing non-printable characters, punctuation and non-alphabetic tokens; normalising unicode to ascii and lowercase.

Split Text

Dataset

Clean data = 150,000 phrase pairs, some very long.

- This is a good number of examples for developing a small translation model. The complexity increases with number of examples, length of phrases, and size of vocabulary.
- Simplify problem slightly to dramatically reduce training time required to fit the model. Reduce dataset to shortest 10,000 phrases.
- Take 9,000 for training and 1,000 to test the fit model.

Train Neural Translation Model

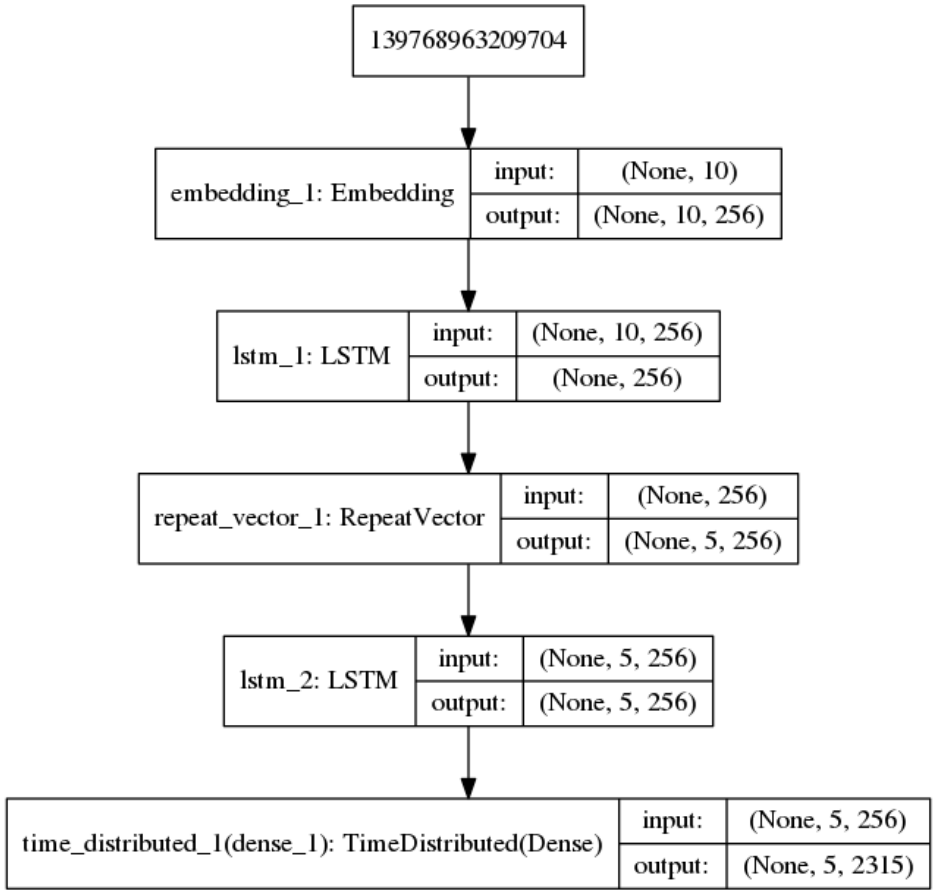


Figure 4: Plot for NMT model graph.

Evaluate Neural Translation Model

```
bojan@soc-LUJGDT6H2-B:~/git/machine-translation$ python evaluate_model.py
src=[ich liebe dich], target=[i love you], predicted=[i love you]
src=[ich sagte du sollst den mund halten], target=[i said shut up], predicted=[i said stop up]
src=[wie geht es eurem vater], target=[hows your dad], predicted=[hows your dad]
src=[das gefallt mir], target=[i like that], predicted=[i like that]
src=[ich gehe immer zu fu], target=[i always walk], predicted=[i will to]
src=[ich konnte nicht gehen], target=[i couldnt walk], predicted=[i cant go]
src=[er ist sehr jung], target=[he is very young], predicted=[he is very young]
src=[versucht es doch einfach], target=[just try it], predicted=[just try it]
src=[sie sind jung], target=[youre young], predicted=[youre young]
src=[er ging surfen], target=[he went surfing], predicted=[he went surfing]

BLEU-1: 0.085682
BLEU-2: 0.284191
BLEU-3: 0.459090
BLEU-4: 0.517571
```

Figure 5: Model evaluation and BLEU scores.

Conclusion

- Machine Translation offers two main different models with very different approaches and results.
- To benefit from both a combination has been suggested.
- Keras as one of the most popular deep learning frameworks in Python offers an easy-to-learn API for machine translation.
- Implementation from scratch can be realised with little effort.
- Although, more complex models would require a serious amount of work, especially when thinking about memory-augmented approaches.

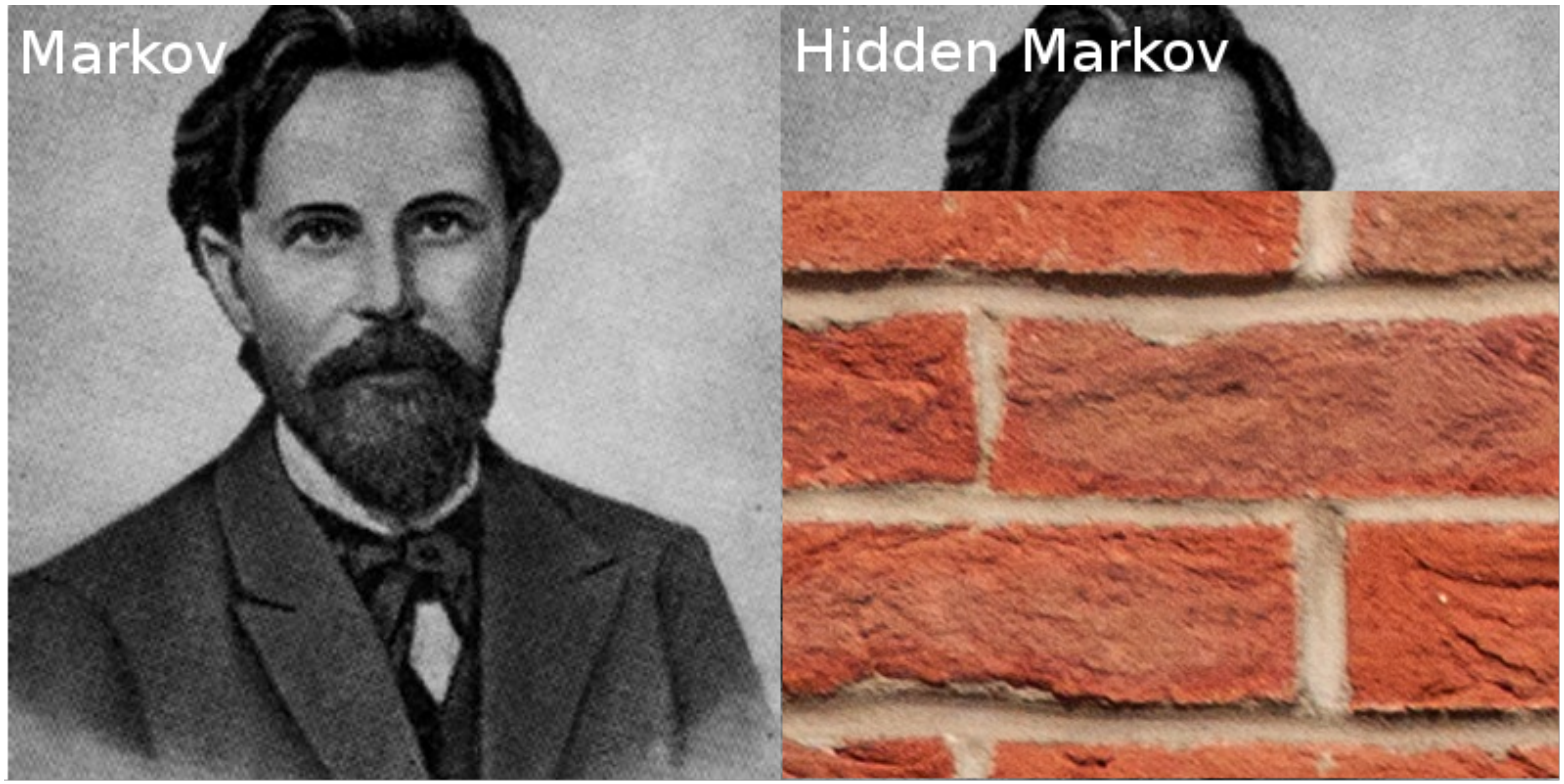
Code

- Neural Machine Translation in keras example: `https://github.com/bozicb/machine-translation`
- Attention-based NMT: `https://github.com/bozicb/attention-translation-keras`
- NMT keras tutorial: `https://github.com/bozicb/nmt-kerasl`
- NMT keras library: `https://nmt-keras.readthedocs.io/en/latest/tutorial.html`
- List of NMT implementations: `https://github.com/bozicb/nmt-list`

Other Resources

- Twitter: @bojan_bozic
- Github: `https://github.com/bozicb`
- M-NMT paper: `https://arxiv.org/abs/1708.02005`
- Keras Sequential Model Guide: `https://keras.io/getting-started/sequential-model-guide/`
- Adam optimiser: `https://arxiv.org/abs/1412.6980v8`

Questions?



Joke first seen in a talk given by Humberto Corona.