Langsim Documentation

Release 1.0

Mark Granroth-Wilding

Oct 25, 2018

Contents

> *Unsupervised Learning of Cross-Lingual Symbol Embeddings Without Parallel Data*
> Mark Granroth-Wilding and Hannu Toivonen (2019)
> In proceedings Society for Computation in Linguistics (SCiL)

This codebase contains the code used to prepare data and train models for this paper. The code is released on Github.

For more information about the paper, including downloadable pre-trained embeddings, see here.

It uses Pimlico. Pimlico pipeline config files can be found in the `pipelines` directory. Most of the code consists of *Pimlico modules (documented here)*.

The code has been cleaned up for release, which involved removing a lot of old code from various experiments carried out over a number of years. Hopefully, I've not removed anything important, but get in touch with Mark if something seems to be missing.

In the paper, the model is called **Xsym**. In this code, it is called **neural_sixgram**.

CHAPTER 1

Getting started

To start using the code, see Pimlico's guide for initializing Pimlico with someone else's code.

In short…

- Download the codebase and extract it.
- Download the bootstrap.py script from Pimlico to the root directory of the codebase.
- In the root directory, run: `python bootstrap.py pipelines/char_embed_corpora.conf`
- Check that the setup has worked:
  - `cd pipelines`
  - `./char_embed_corpora.conf status`
  - Pimlico should do some initial setup and then show a long list of modules in the pipeline
- Delete `bootstrap.py`

CHAPTER 2

Pipelines

There are two pipelines. These cover the corruption experiment and the main model training described in the paper.

In addition to this, if you want to reproduce everything we did, you'll need to preprocess the data for low-resourced Uralic languages to clean it up. That process is implemented and documented in a separate codebase, which also uses Pimlico.

2.1 char_embed_corpora

Main model training pipeline.

This pipeline loads a variety of corpora and trains Xsym on them. It produces all the models described in the paper. To train on these corpora, you'll need to download them and then update the paths in the `[vars]` section to point to their locations.

There are two slightly different implementations of the training code, found in the Pimlico modules *`neural_sixgram`* and *`neural_sixgram2`*. If you're training the model yourself, you should use the more recent and more efficient *`neural_sixgram2`*.

The pipeline also includes training on some language pairs not reported in the paper.

2.2 char_embed_corrupt

Language corruption experiments to test Xsym's robustness to different types of noise.

This pipeline implements the language corruption experiments reported in the paper. It takes real language data (Finnish forum posts) and applies random corruptions to it, training Xsym on uncorrupted and corrupted pairs.

CHAPTER 3

Corpora

3.1 Ylilauta

Finnish forum posts.

- Download.

3.2 Estonian Reference Corpus

Corpus of written Estonian from a variety of sources. Here we use just the subsets: `tasakaalus_ajalehed` and `foorumid_lausestatud`.

- Corpus
- Forum post subset

3.3 Danish Wikipedia dump

Text dump of Danish Wikipedia.

- Download

3.4 Europarl

The Europarl corpus of transcripts from the European Parliament.

Download the full source release. We use the Swedish, Spanish and Portuguese parts.

- Homepage

3.5 Multilingual Resource Collection of the University of Helsinki Language Corpus Server (UHLCS)

Data used to be available from the homepage, but is now available through the CSC. You'll need to request access to the specific language datasets used.

The data you get is messy, in inconsistent formats and encodings. See the code distributed separately for how to preprocess this and get it into a useable textual form, which we use below.

- UHLCS homepage
- CSC
- My code for preparing the corpora

CHAPTER 4

Documentation

4.1 Pimlico modules

Pimlico modules for symbol embedding experiments. These are used in the Pimlico pipelines in the `pipelines/` directory.

4.1.1 Fake language tools

Some tools for generating fake language data.

This performs the language corruption used in the paper to test the robustness of the Xsym model.

Corrupt text

Path	langsim.modules.fake_language.corrupt
Executable	yes

Introduce random noise into a corpus.

The input corpus is expected to be character-level encoded integer indexed text. (You could also run it on word-level encoded data, but the results might be odd.)

Produces a new corpus with a new character vocabulary, which might not be identical to the input vocabulary, depending on the options. E.g. some characters might be removed or added.

If a token called 'OOV' is found in the vocabulary, it will never be subject to a mapping or mapped to.

Types of noise, with corresponding parameters:

- Random character substitutions: randomly sample a given proportion of characters and choose a character at random from the unigram distribution of the input corpus to replace each with

  - `char_subst_prop`: proportion of characters (tokens) to sample for substitution. Use 0 to disable this corruption

- Systematic character mapping: perform a systematic substitution throughout the corpus of a particular character A (randomly chosen from input vocab) for another B (randomly chosen from output vocab). This means that the resulting Bs are indistinguishable from those that were Bs in the input. A is removed from the output vocab, since it is never used now. When multiple mappings are chosen, it is not checked that they have different Bs.

  A number of characters is chosen using frequencies so that the expected proportion of tokens affected is at least the given parameter. Since the resulting expected proportion of tokens may be higher due to the sampling of characters, the actual expected proportion is output among the corruption parameters as `actual_char_subst_prop`.

  - `char_map_prop`: proportion of characters (types) in input vocab to apply a mapping to. Use 0 to disable this corruption

- Split characters: choose a set of characters. For each A invent a new character B and map half of its occurrences to B, leaving half as they were. Each of these results in adding a brand new unicode character to the output vocab

  As with `char_map_prop`, a number of characters is chosen using frequencies so that the expected proportion of tokens affected is at least the given parameter. Since the resulting expected proportion of tokens may be higher due to the sampling of characters, the actual expected proportion is output among the corruption parameters as `actual_char_split_prop`.

  - `char_split_prop`: proportion of characters (types) to apply this splitting to

Inputs

Name	Type(s)
corpus	TarredCorpus<IntegerListsDocumentType>
vocab	`Dictionary`
frequencies	`NumpyArray`

Outputs

Name	Type(s)
corpus	`IntegerListsDocumentTypeTarredCorpus`
vocab	`Dictionary`
mappings	`NamedFile()`
close_pairs	`NamedFile()`
corruption_params	`NamedFile()`

Options

Name	Description	Type
char_map_prop	Proportion of character types in input vocab to apply a random mapping to another character to. Default: 0	float
char_split_prop	Proportion of character types in input vocab to apply splitting to. Default: 0	float
char_subst_prop	Proportion of characters to sample for random substitution. Default: 0	float

Inspect corrupted text

Path	langsim.modules.fake_language.inspect
Executable	yes

Display corrupted and uncorrupted texts alongside one another

For observing the output of the corruption process, which otherwise is just a load of integer-encoded data.

Inputs

Name	Type(s)
corpus1	TarredCorpus<IntegerListsDocumentType>
vocab1	`Dictionary`
corpus2	TarredCorpus<IntegerListsDocumentType>
vocab2	`Dictionary`

Outputs

Name	Type(s)
inspect	`RawTextDocumentTypeTarredCorpus`

4.1.2 Input readers

Est Ref normalization

Path	langsim.modules.input.est_ref_normalize
Executable	yes

Special normalization routine for Estonian Reference Corpus.

Splits up sentences into separate lines. This is easy to do, since the corpus puts a double space between sentences. There are also double spaces in other places, so we only split on double spaces after punctuation. Other double spaces are removed.

We also lower-case the whole corpus.

Inputs

Name	Type(s)
corpus	TarredCorpus<TextDocumentType>

Outputs

Name	Type(s)
corpus	`RawTextDocumentTypeTarredCorpus`

Options

Name	Description	Type
forum	Set to T for processing the forum data, which is slightly different to the newspaper data	bool

Europarl corpus reader

Path	langsim.modules.input.europarl
Executable	no

This is an input module. It takes no pipeline inputs and is used to read in data

Inputs

No inputs

Outputs

Name	Type(s)
corpus	`OutputType`

Options

Name	Description	Type
files	(required) Comma-separated list of absolute paths to files to include in the collection. Paths may include globs. Place a '?' at the start of a filename to indicate that it's optional. You can specify a line range for the file by adding ':X-Y' to the end of the path, where X is the first line and Y the last to be included. Either X or Y may be left empty. (Line numbers are 1-indexed.)	comma-separated list of (line range-limited) file paths
exclude	A list of files to exclude. Specified in the same way as *files* (except without line ranges). This allows you to specify a glob in *files* and then exclude individual files from it (you can use globs here too)	comma-separated list of strings
encoding_errors	What to do in the case of invalid characters in the input while decoding (e.g. illegal utf-8 chars). Select 'strict' (default), 'ignore', 'replace'. See Python's str.decode() for details	string
encoding	Encoding to assume for input files. Default: utf8	string

Ylilauta VRT files

Path	langsim.modules.input.ylilauta
Executable	yes

Input reader for Ylilauta corpus.

Based on standard VRT text collection module, with a small amount of special processing added for Ylilauta.

See also:

`pimlico.modules.input.text_annotations.vrt_text`: Reading text from VRT files.

This is an input module. It takes no pipeline inputs and is used to read in data

Inputs

No inputs

Outputs

Name	Type(s)
corpus	`YlilautaOutputType`

Options

Name	Description	Type
files	(required) Comma-separated list of absolute paths to files to include in the collection. Paths may include globs. Place a '?' at the start of a filename to indicate that it's optional. You can specify a line range for the file by adding ':X-Y' to the end of the path, where X is the first line and Y the last to be included. Either X or Y may be left empty. (Line numbers are 1-indexed.)	comma-separated list of (line range-limited) file paths
exclude	A list of files to exclude. Specified in the same way as *files* (except without line ranges). This allows you to specify a glob in *files* and then exclude individual files from it (you can use globs here too)	comma-separated list of strings
encoding_errors	What to do in the case of invalid characters in the input while decoding (e.g. illegal utf-8 chars). Select 'strict' (default), 'ignore', 'replace'. See Python's str.decode() for details	string
encoding	Encoding to assume for input files. Default: utf8	string

4.1.3 Symbol embedding methods

Neural network-based symbol (phoneme/character) representation learning techniques that work by applying the distributional hypothesis cross-lingually and simultaneously learning representations for both languages.

Some ways of doing this work better than others. The best method appears to be `neural_sixgram2`, which is now the only one implemented here. It takes into account a relatively broad context of the symbols, and seems to be fairly robust across language pairs.

Corruption results

Path	langsim.modules.local_lm.corruption_results
Executable	yes

Collect results from the corruption experiments, including models trained on corrupted corpora, and analyse them.

Inputs

Name	Type(s)
corruption_params	`list` of A file collection containing at least one file (or a given specific number). No constraint is put on the name of the file(s). Typically, the module will just use whatever the first file(s) in the collection is
models	`list` of `KerasModelBuilderClass`
vocab1s	`list` of `Dictionary`
vocab2s	`list` of `Dictionary`
mapped_pairs	`list` of A file collection containing at least one file (or a given specific number). No constraint is put on the name of the file(s). Typically, the module will just use whatever the first file(s) in the collection is

Outputs

Name	Type(s)
analysis	`NamedFile()`
files	`UnnamedFileCollection`

Learned embedding analysis

Path	langsim.modules.local_lm.embed_anal
Executable	yes

Various analyses thrown together for including things in a paper.

To simplify things, we assume for now that there are exactly two languages (vocabs, corpora). We could generalize this later, but for now it makes the code much easier and we only do this for the paper.

Inputs

Name	Type(s)
model	`NeuralSixgramKerasModel`
vocabs	`list` of `Dictionary`
frequencies	`list` of `NumpyArray`

Outputs

Name	Type(s)
analysis	`NamedFile()`
pairs	`NamedFile()`

Options

Name	Description	Type
oov	If given, look for this special token in each vocabulary which represents OOVs. These are not filtered out, even if they are rare	string
lang_names	(required) Comma-separated list of language IDs to use in output	comma-separated list of strings
min_token_prop	Minimum frequency, as a proportion of tokens, that a character in the vocabulary must have to be shown in the charts	float

Embeddings from model

Path	langsim.modules.local_lm.embeddings_from_model
Executable	yes

Simple module to extract the trained embeddings from a model stored by the training process, which can then be used in a generic way and output to generic formats.

Inputs

Name	Type(s)
model	`NeuralSixgramKerasModel`
vocabs	`list` of `Dictionary`
frequencies	`list` of `NumpyArray`

Outputs

Name	Type(s)
embeddings	`Embeddings`

Options

Name	Description	Type
lang_names	(required) Comma-separated list of language IDs to use in output	comma-separated list of strings

Language-specific embeddings

Path	langsim.modules.local_lm.lang_embeddings
Executable	yes

Separate out the embeddings belonging to the two languages, identified by prefixes on the words.

It's assumed that all embeddings for language "X" have words of the form "X:word".

This only works currently for cases where there are exactly two languages.

Inputs

Name	Type(s)
embeddings	`Embeddings`

Outputs

Name	Type(s)
lang1_embeddings	`Embeddings`
lang2_embeddings	`Embeddings`

Options

Name	Description	Type
lang1	Prefixes for language 1. If not given, language 1 is taken to be whichever appears first in the vocabulary	string

Neural sixgram (Xsym) trainer, v1

Path	langsim.modules.local_lm.neural_sixgram
Executable	yes

A special kind of six-gram model that combines 1-3 characters on the left with 1-3 characters on the right to learn unigram, bigram and trigram representations.

This is one of the most successful representation learning methods among those here. It's also very robust across language pairs and different sizes of dataset. It's therefore the model that I've opted to use in subsequent work that uses the learned representations.

Inputs

Name	Type(s)
vocabs	`list` of `Dictionary`
corpora	`list` of TarredCorpus<IntegerListsDocumentType>
frequencies	`list` of `NumpyArray`

Outputs

Name	Type(s)
model	`KerasModelBuilderClass`

Options

Name	Description	Type
embedding_size	Number of dimensions in the hidden representation. Default: 200	int
plot_freq	Output plots to the output directory while training is in progress. This slows down training if it's done very often. Specify how many batches to wait between each plot. Fewer means you get a finer grained picture of the training process, more means training goes faster. 0 (default) turns off plotting	int
context_weights	Coefficients that specify the relative frequencies with which each of the different lengths of contexts (1, 2 and 3) will be used in training examples. For each sample, a pair context lengths is selected at random. Six coefficients specify the weights given to (1,1), (1,2), (1,3), (2,2), (2,3) and (3,3). The opposite orderings have the same probability. By default, they are uniformly sampled ('1,1,1,1,1,1'), but you may adjust their relative frequencies to put more weight on some lengths than others. The first 6 values are the starting weights. After that, you may specify sets of 7 values: num_epochs, weight1, weight2, ... The weights at any point will transition smoothly (linearly) from the previous 6-tuple to the next, arriving at the epoch number given (i.e. 1=start of epoch 1 / end of first epoch). You may use float epoch numbers, e.g. 0.5	<function context_weights at 0x7f3b52ef3050>
composition2_layers	Number and size of layers to use to combine pairs of characters, given as a list of integers. The final layer must be the same size as the embeddings, so is not included in this list	comma-separated list of ints
epochs	Max number of training epochs. Default: 5	int
predictor_layers	Number and size of layers to use to take a pair of vectors and say whether they belong beside each other. Given as a list of integers. Doesn't include the final projection to a single score	comma-separated list of ints
limit_training	Limit training to this many batches. Default: no limit	int
l2_reg	L2 regularization to apply to all layers' weights. Default: 0.	float
unit_norm	If true, enforce a unit norm constraint on the learned embeddings. Default: false	bool
word_internal	Only train model on word-internal sequences. Word boundaries will be included, but no sequences spanning over word boundaries	bool
dropout	Dropout to apply to embeddings during training. Default: 0.3	float
oov	If given, use this special token in each vocabulary to represent OOVs. Otherwise, they are represented by an index added at the end of each vocabulary's indices	string
word_boundary	If using word_internal, use this character (which must be in the vocabulary) to split words. Default: space	<type 'unicode'>
composition3_layers	Number and size of layers to use to combine triples of characters, given as a list of integers. The final layer must be the same size as the embeddings, so is not included in this list	comma-separated list of ints
store_all	Store updated representations from every epoch, even if the validation loss goes up. The default behaviour is to only store the parameters with best validation loss, but for these purposes we probably want to set this to T most of the time. (Defaults to F for backwards compatibility)	bool
composition_dropout	Dropout to apply to composed representation during training. Default: same as dropout	float
batch	Training batch size. Default: 100	int
sim_freq	How often (in batches) to compute the similarity of overlapping phonemes between the languages. -1 (default) means never, 0 means once at the start of each epoch	int
corpus_offset	To avoid training on parallel data, in the case where the input corpora happen to be parallel, jump forward in the second corpus by this number of utterances, putting the skipping utterances at the end instead. Default: 10k utterances	int

Neural sixgram (Xsym) trainer, v2

Path	langsim.modules.local_lm.neural_sixgram2
Executable	yes

A special kind of six-gram model that combines 1-3 characters on the left with 1-3 characters on the right to learn unigram, bigram and trigram representations.

This is one of the most successful representation learning methods among those here. It's also very robust across language pairs and different sizes of dataset. It's therefore the model that I've opted to use in subsequent work that uses the learned representations.

This is a new version of the code for the model training. It will include random restarts and early stopping using the new validation criterion. I've moved to a new version so that I can get rid of old things from experiments with different types of models and clean up the code. The old version was used to measure the validity of the validation criterion. From now on, I'm using the validation criterion in earnest.

I'm now changing all default parameters to those use in the submitted paper and removing some parameters for features that no longer need to be parameterized.

Note: A note on using GPUs

We use Keras to train. If you're using the tensorflow backend (which is what is assumed by this module's dependencies) and you want to use GPUs, you'll need to install the GPU version of Tensorflow, not just "tensorflow", which will be installed during dependency resolution. Try this (changing the virtualenv directory name if you're not using the default):

```
./pimlico/lib/virtualenv/default/bin/pip install --upgrade tensorflow-gpu
```

Note: *Changed 12.09.18*: this module takes prepared positive sample data as input instead of doing the preparation (random shuffling, etc) during training. I found a bug that meant that we weren't training on the full datasets, so training actually takes much longer than it seemed. It's therefore important not to waste time redoing data processing on each training epoch.

Some pipelines that were written before this change will no longer work, but they're quite simple to fix. Add an extra data preparation module before the training module, taking the inputs and parameters from the training module as appropriate (and removing some of them from there).

Inputs

Name	Type(s)
vocabs	`list` of `Dictionary`
samples	`NeuralSixgramTrainingData`

Outputs

Name	Type(s)
model	`NeuralSixgramKerasModel`

Options

Name	Description	Type
composition3_layers	Number and size of layers to use to combine triples of characters, given as a list of integers. The final layer must be the same size as the embeddings, so is not included in this list. Default: nothing, i.e. linear transformation	comma-separated list of ints
embedding_size	Number of dimensions in the hidden representation. Default: 30	int
composition_dropout	Dropout to apply to composed representation during training. Default: 0.01	float
predictor_layers	Number and size of layers to use to take a pair of vectors and say whether they belong beside each other. Given as a list of integers. Doesn't include the final projection to a single score. Default: 30 (single hidden layer)	comma-separated list of ints
dropout	Dropout to apply to embeddings during training. Default: 0.1	float
plot_freq	Output plots to the output directory while training is in progress. This slows down training if it's done very often. Specify how many batches to wait between each plot. Fewer means you get a finer grained picture of the training process, more means training goes faster. -1 turns off plotting. 0 (default) means once at the start/end of each epoch	int
patience	Early stopping patience. Number of epochs with no improvement after which training will be stopped. Default: 2	int
batch	Training batch size in training samples (pos-neg pairs). Default: 1000	int
composition2_layers	Number and size of layers to use to combine pairs of characters, given as a list of integers. The final layer must be the same size as the embeddings, so is not included in this list. Default: nothing, i.e. linear transformation	comma-separated list of ints
restarts	How many random restarts to perform. Each time, the model is randomly re-initialized from scratch. All models are saved and the one with the best value of the validation criterion is stored as the output. Default: 1, just train once	int
epochs	Max number of training epochs. Default: 10	int
split_epochs	Normal behaviour is to iterate over the full dataset once in each epoch, generating random negative samples to accompany it. Early stopping is done using the validation metric over the learned representations after each epoch. With larger datasets, this may mean waiting too long before we start measuring the validation metric. If split_epochs > 1, one epoch involves 1/split_epochs of the data. The following epoch continues iterating over the dataset, so all the data gets used, but the early stopping checks are performed split_epochs times in each iteration over the dataset	int
sim_freq	How often (in batches) to compute the similarity of overlapping phonemes between the languages. -1 (default) means never, 0 means once at the start of each epoch. If input mapped_pairs is given, the similarity is computed between these pairs; otherwise we use any identical pairs that exist between the vocabularies	int
limit_training	Limit training to this many batches. Default: no limit	int
validation	Number of samples to hold out as a validation set for training. Simply taken from the start of the corpus. Rounded to the nearest number of batches	int
unit_norm	If true, enforce a unit norm constraint on the learned embeddings. Default: true	bool

Neural sixgram samples prep

Path	langsim.modules.local_lm.neural_sixgram_samples
Executable	yes

Prepare positive samples for neural sixgram training data.

Instead of doing random shuffling, etc, on the fly while training, which takes quite a lot of time, we do it once before and just iterate over the result at training time.

The output is then used by *neural_sixgram2* to train the Xsym model.

Inputs

Name	Type(s)
vocabs	`list` of `Dictionary`
corpora	`list` of TarredCorpus<IntegerListsDocumentType>
frequencies	`list` of `NumpyArray`

Outputs

Name	Type(s)
samples	`NeuralSixgramTrainingData`

Options

Name	Description	Type
cross_sentences	By default, the sliding window passed over the corpus stops at the end of a sentence (or whatever sequence division is in the input data) and starts again at the start of the next. Instead, join all sequences within a document into one long sequence and pass the sliding window over that	bool
oov	If given, use this special token in each vocabulary to represent OOVs. Otherwise, they are represented by an index added at the end of each vocabulary's indices	string
shuffle_window	We simulate shuffling the data by reading samples into a buffer and taking them randomly from there. This is the size of that buffer. A higher number shuffles more, but makes data preparation slower	int
corpus_offset	To avoid training on parallel data, in the case where the input corpora happen to be parallel, jump forward in the second corpus by this number of utterances, putting the skipping utterances at the end instead. Default: 10k utterances	int

Plots of neural sixgram models

Path	langsim.modules.local_lm.plot
Executable	yes

Produces various plots to help with analysing the results of training a neural_sixgram model.

Note that this used to be designed to support other model types, but I'm now cleaning up and only supporting neural_sixgram2.

Inputs

Name	Type(s)
model	KerasModelBuilderClass
vocabs	list of Dictionary
corpora	list of TarredCorpus<IntegerListsDocumentType>
frequencies	list of NumpyArray

Outputs

Name	Type(s)
output	PimlicoDatatype

Options

Name	Description	Type
distance	Distance metric to use	'eucl', 'dot', 'cos', 'man' or 'sig_kern'
num_pairs	Number of most frequent character pairs to show on the chart (passed through the composition function to get their representation)	int
min_token_prop	Minimum frequency, as a proportion of tokens, that a character in the vocabulary must have to be shown in the charts	float
lang_names	(required) Comma-separated list of language IDs to use in output	comma-separated list of strings

Store in TSV format

Path	langsim.modules.local_lm.store_tsv
Executable	yes

Takes embeddings stored in the default format used within Pimlico pipelines (see Embeddings) and stores them as TSV files.

These are suitable as input to the Tensorflow Projector.

Like the built-in store_tsv module, but includes some additional language information in the metadata to help with visualization.

Inputs

Name	Type(s)
embeddings	Embeddings

Outputs

Name	Type(s)
embeddings	`TSVVecFiles`

Validation criterion correlation

Path	langsim.modules.local_lm.val_crit_correlation
Executable	yes

Compute correlation between the validation criterion and the retrieval of known correspondences. See the paper for more details.

Inputs

Name	Type(s)
models	`list` of `KerasModelBuilderClass`

Outputs

Name	Type(s)
metrics	`NamedFile()`
final_metrics	`NamedFile()`
correlations	`NamedFile()`

4.2 langsim package

4.2.1 Subpackages

langsim.datatypes package

Submodules

langsim.datatypes.neural_sixgram module

Module contents

4.2.2 Module contents

Python Module Index

d

l

m

Index

L