LM-LSTM-CRF Documentation

Liyuan Liu, Frank Xu, Jingbo Shang

Nov 08, 2018

Notes

Check Our New NER Toolkit

- **Inference**:
  - LightNER: inference w. models pre-trained / trained w. *any* following tools, *efficiently*.
- **Training**:
  - LD-Net: train NER models w. efficient contextualized representations.
  - VanillaNER: train vanilla NER models w. pre-trained embedding.
- **Distant Training**:
  - AutoNER: train NER models w.o. line-by-line annotations and get competitive performance.

This project provides high-performance character-aware sequence labeling tools, including [Training](#usage), [Evaluation](#evaluation) and [Prediction](#prediction).

Details about LM-LSTM-CRF can be accessed here, and the implementation is based on the PyTorch library.

CHAPTER 1

model package

1.1 Submodules

1.2 model.crf module

class model.crf.**CRFDecode_vb**(*tagset_size*, *start_tag*, *end_tag*, *average_batch=True*)

Bases: object

Batch-mode viterbi decode

Parameters

- **tagset_size** – target_set_size
- **start_tag** – ind for <start>
- **end_tag** – ind for <pad>
- **average_batch** – whether average the loss among batch

decode(*scores*, *mask*)

Find the optimal path with viterbe decode

Parameters

- **scores** (*size seq_len, bat_size, target_size_from, target_size_to*) – crf scores
- **mask** (*seq_len, bat_size*) – mask for padding

Returns decoded sequence (size seq_len, bat_size)

class model.crf.**CRFLoss_gd**(*tagset_size*, *start_tag*, *end_tag*, *average_batch=True*)

Bases: torch.nn.modules.module.Module

loss for greedy decode loss, i.e., although its for CRF Layer, we calculate the loss as

$$\sum_{j=1}^{n} \log(p(\hat{y}_{j+1}|z_{j+1}, \hat{y}_j))$$

instead of

$$\sum_{j=1}^{n} \log(\phi(\hat{y}_{j-1}, \hat{y}_j, \mathbf{z}_j)) - \log(\sum_{\mathbf{y}' \in \mathbf{Y}(\mathbf{Z})} \prod_{j=1}^{n} \phi(y'_{j-1}, y'_j, \mathbf{z}_j))$$

Parameters

- **tagset_size** – target_set_size
- **start_tag** – ind for <start>
- **end_tag** – ind for <pad>
- **average_batch** – whether average the loss among batch

forward(*scores*, *target*, *current*)

Parameters

- **scores** (*Word_Seq_len, Batch_size, target_size_from, target_size_to*) – crf scores
- **target** (*Word_Seq_len, Batch_size*) – golden list
- **current** (*Word_Seq_len, Batch_size*) – current state

Returns crf greedy loss

class model.crf.**CRFLoss_vb**(*tagset_size*, *start_tag*, *end_tag*, *average_batch=True*)

Bases: torch.nn.modules.module.Module

loss for viterbi decode

$$\sum_{j=1}^{n} \log(\phi(\hat{y}_{j-1}, \hat{y}_j, \mathbf{z}_j)) - \log(\sum_{\mathbf{y}' \in \mathbf{Y}(\mathbf{Z})} \prod_{j=1}^{n} \phi(y'_{j-1}, y'_j, \mathbf{z}_j))$$

Parameters

- **tagset_size** – target_set_size
- **start_tag** – ind for <start>
- **end_tag** – ind for <pad>
- **average_batch** – whether average the loss among batch

forward(*scores*, *target*, *mask*)

Parameters

- **scores** (*seq_len, bat_size, target_size_from, target_size_to*) – crf scores
- **target** (*seq_len, bat_size, 1*) – golden state
- **mask** (*size seq_len, bat_size*) – mask for padding

Returns loss

class model.crf.**CRFRepack**(*tagset_size*, *if_cuda*)
Bases: object

Packer for word level model

Parameters

- **tagset_size** – target_set_size
- **if_cuda** – whether use GPU

convert_for_eval(*target*)
convert target to original decoding

Parameters **target** – input labels used in training

Returns output labels used in test

repack_gd(*feature*, *target*, *current*)
packer for greedy loss

Parameters

- **feature** (*Seq_len, Batch_size*) – input feature
- **target** (*Seq_len, Batch_size*) – output target
- **current** (*Seq_len, Batch_size*) – current state

Returns feature (Seq_len, Batch_size), target (Seq_len * Batch_size), current (Seq_len * Batch_size, 1, 1)

repack_vb(*feature*, *target*, *mask*)
packer for viterbi loss

Parameters

- **feature** (*Seq_len, Batch_size*) – input feature
- **target** (*Seq_len, Batch_size*) – output target
- **mask** (*Seq_len, Batch_size*) – padding mask

Returns feature (Seq_len, Batch_size), target (Seq_len, Batch_size), mask (Seq_len, Batch_size)

class model.crf.**CRFRepack_WC**(*tagset_size*, *if_cuda*)
Bases: object

Packer for model with char-level and word-level

Parameters

- **tagset_size** – target_set_size
- **if_cuda** – whether use GPU

convert_for_eval(*target*)
convert for eval

Parameters **target** – input labels used in training

Returns output labels used in test

repack_vb(*fc_feature*, *fc_position*, *bc_feature*, *bc_position*, *word_feature*, *target*, *mask*, *batch_len*)
packer for viterbi loss

Parameters

- **fc_feature** (*Char_Seq_len, Batch_size*) – forward_char input feature
- **fc_position** (*Word_Seq_len, Batch_size*) – forward_char input position
- **bc_feature** (*Char_Seq_len, Batch_size*) – backward_char input feature
- **bc_position** (*Word_Seq_len, Batch_size*) – backward_char input position
- **word_feature** (*Word_Seq_len, Batch_size*) – input word feature
- **target** (*Seq_len, Batch_size*) – output target
- **mask** (*Word_Seq_len, Batch_size*) – padding mask
- **batch_len** (*Batch_size, 2*) – length of instances in one batch

Returns f_f (Char_Reduced_Seq_len, Batch_size), f_p (Word_Reduced_Seq_len, Batch_size), b_f (Char_Reduced_Seq_len, Batch_size), b_p (Word_Reduced_Seq_len, Batch_size), w_f (size Word_Seq_Len, Batch_size), target (Reduced_Seq_len, Batch_size), mask (Word_Reduced_Seq_len, Batch_size)

class model.crf.**CRF_L**(*hidden_dim*, *tagset_size*, *if_bias=True*)

Bases: torch.nn.modules.module.Module

Conditional Random Field (CRF) layer. This version is used in Ma et al. 2016, has more parameters than CRF_S

Parameters

- **hidden_dim** – input dim size
- **tagset_size** – target_set_size
- **if_biase** – whether allow bias in linear trans

forward(*feats*)

Parameters **feats** (*batch_size, seq_len, hidden_dim*) – input score from previous layers

Returns output from crf layer (batch_size, seq_len, tag_size, tag_size)

rand_init()

random initialization

class model.crf.**CRF_S**(*hidden_dim*, *tagset_size*, *if_bias=True*)

Bases: torch.nn.modules.module.Module

Conditional Random Field (CRF) layer. This version is used in Lample et al. 2016, has less parameters than CRF_L.

Parameters

- **hidden_dim** – input dim size
- **tagset_size** – target_set_size
- **if_biase** – whether allow bias in linear trans

forward(*feats*)

Parameters **feats** (*batch_size, seq_len, hidden_dim*) – input score from previous layers

Returns output from crf layer ((batch_size * seq_len), tag_size, tag_size)

rand_init()

random initialization

1.3 model.evaluator module

class model.evaluator.**eval_batch**(*packer*, *l_map*)
Bases: object

Base class for evaluation, provide method to calculate f1 score and accuracy

Parameters

- **packer** – provide method to convert target into original space [TODO: need to improve]
- **l_map** – dictionary for labels

acc_score()
calculate accuracy score based on statics

calc_acc_batch(*decoded_data*, *target_data*)
update statics for accuracy

Parameters

- **decoded_data** (*batch_size, seq_len*) – prediction sequence
- **target_data** (*batch_size, seq_len*) – ground-truth

calc_f1_batch(*decoded_data*, *target_data*)
update statics for f1 score

Parameters

- **decoded_data** (*batch_size, seq_len*) – prediction sequence
- **target_data** (*batch_size, seq_len*) – ground-truth

eval_instance(*best_path*, *gold*)
update statics for one instance

Parameters

- **best_path** (*seq_len*) – predicted
- **gold** (*seq_len*) – ground-truth

f1_score()
calculate f1 score based on statics

reset()
re-set all states

class model.evaluator.**eval_w**(*packer*, *l_map*, *score_type*)
Bases: *model.evaluator.eval_batch*

evaluation class for word level model (LSTM-CRF)

Parameters

- **packer** – provide method to convert target into original space [TODO: need to improve]
- **l_map** – dictionary for labels
- **score_type** – use f1score with using 'f'

calc_score(*ner_model*, *dataset_loader*)
calculate score for pre-selected metrics

Parameters

- **ner_model** – LSTM-CRF model
- **dataset_loader** – loader class for test set

class model.evaluator.**eval_wc**(*packer*, *l_map*, *score_type*)

Bases: *model.evaluator.eval_batch*

evaluation class for LM-LSTM-CRF

Parameters

- **packer** – provide method to convert target into original space [TODO: need to improve]
- **l_map** – dictionary for labels
- **score_type** – use f1score with using 'f'

calc_score(*ner_model*, *dataset_loader*)

calculate score for pre-selected metrics

Parameters

- **ner_model** – LM-LSTM-CRF model
- **dataset_loader** – loader class for test set

1.4 model.highway module

class model.highway.**hw**(*size*, *num_layers=1*, *dropout_ratio=0.5*)

Bases: torch.nn.modules.module.Module

Highway layers

Parameters

- **size** – input and output dimension
- **dropout_ratio** – dropout ratio

forward(*x*)

update statics for f1 score

Parameters **x** (*ins_num, hidden_dim*) – input tensor

Returns output tensor (ins_num, hidden_dim)

rand_init()

random initialization

1.5 model.lm_lstm_crf module

class model.lm_lstm_crf.**LM_LSTM_CRF**(*tagset_size*, *char_size*, *char_dim*, *char_hidden_dim*, *char_rnn_layers*, *embedding_dim*, *word_hidden_dim*, *word_rnn_layers*, *vocab_size*, *dropout_ratio*, *large_CRF=True*, *if_highway=False*, *in_doc_words=2*, *highway_layers=1*)

Bases: torch.nn.modules.module.Module

LM_LSTM_CRF model

Parameters

- **tagset_size** – size of label set
- **char_size** – size of char dictionary
- **char_dim** – size of char embedding
- **char_hidden_dim** – size of char-level lstm hidden dim
- **char_rnn_layers** – number of char-level lstm layers
- **embedding_dim** – size of word embedding
- **word_hidden_dim** – size of word-level blstm hidden dim
- **word_rnn_layers** – number of word-level lstm layers
- **vocab_size** – size of word dictionary
- **dropout_ratio** – dropout ratio
- **large_CRF** – use CRF_L or not, refer model.crf.CRF_L and model.crf.CRF_S for more details
- **if_highway** – use highway layers or not
- **in_doc_words** – number of words that occurred in the corpus (used for language model prediction)
- **highway_layers** – number of highway layers

forward(*forw_sentence*, *forw_position*, *back_sentence*, *back_position*, *word_seq*, *hidden=None*)

Parameters

- **forw_sentence** (*char_seq_len, batch_size*) – char-level representation of sentence
- **forw_position** (*word_seq_len, batch_size*) – position of blank space in char-level representation of sentence
- **back_sentence** (*char_seq_len, batch_size*) – char-level representation of sentence (inverse order)
- **back_position** (*word_seq_len, batch_size*) – position of blank space in inversed char-level representation of sentence
- **word_seq** (*word_seq_len, batch_size*) – word-level representation of sentence
- **hidden** – initial hidden state

Returns crf output (word_seq_len, batch_size, tag_size, tag_size), hidden

load_pretrained_word_embedding(*pre_word_embeddings*)

load pre-trained word embedding

Parameters **pre_word_embeddings** (*self.word_size, self.word_dim*) – pre-trained embedding

rand_init(*init_char_embedding=True*, *init_word_embedding=False*)

random initialization

Parameters

- **init_char_embedding** – random initialize char embedding or not
- **init_word_embedding** – random initialize word embedding or not

rand_init_embedding()
random initialize char-level embedding

set_batch_seq_size(*sentence*)
set batch size and sequence length

set_batch_size(*bsize*)
set batch size

word_pre_train_backward(*sentence*, *position*, *hidden=None*)
output of backward language model

Parameters

- **sentence** (*char_seq_len, batch_size*) – char-level representation of sentence (inverse order)
- **position** (*word_seq_len, batch_size*) – position of blank space in inversed char-level representation of sentence
- **hidden** – initial hidden state

Returns language model output (word_seq_len, in_doc_word), hidden

word_pre_train_forward(*sentence*, *position*, *hidden=None*)
output of forward language model

Parameters

- **sentence** (*char_seq_len, batch_size*) – char-level representation of sentence
- **position** (*word_seq_len, batch_size*) – position of blank space in char-level representation of sentence
- **hidden** – initial hidden state

Returns language model output (word_seq_len, in_doc_word), hidden

1.6 model.lstm_crf module

class model.lstm_crf.**LSTM_CRF**(*vocab_size*, *tagset_size*, *embedding_dim*, *hidden_dim*, *rnn_layers*, *dropout_ratio*, *large_CRF=True*)
Bases: torch.nn.modules.module.Module

LSTM_CRF model

Parameters

- **vocab_size** – size of word dictionary
- **tagset_size** – size of label set
- **embedding_dim** – size of word embedding
- **hidden_dim** – size of word-level blstm hidden dim
- **rnn_layers** – number of word-level lstm layers
- **dropout_ratio** – dropout ratio
- **large_CRF** – use CRF_L or not, refer model.crf.CRF_L and model.crf.CRF_S for more details

forward(*sentence*, *hidden=None*)

Parameters

- **sentence** (*word_seq_len, batch_size*) – word-level representation of sentence
- **hidden** – initial hidden state

Returns crf output (word_seq_len, batch_size, tag_size, tag_size), hidden

load_pretrained_embedding(*pre_embeddings*)
load pre-trained word embedding

Parameters **pre_word_embeddings** (*self.word_size, self.word_dim*) – pre-trained embedding

rand_init(*init_embedding=False*)
random initialization

Parameters **init_embedding** – random initialize embedding or not

rand_init_embedding()

rand_init_hidden()
random initialize hidden variable

set_batch_seq_size(*sentence*)
set batch size and sequence length

set_batch_size(*bsize*)
set batch size

1.7 model.ner_dataset module

class model.ner_dataset.**CRFDataset**(*data_tensor*, *label_tensor*, *mask_tensor*)
Bases: torch.utils.data.dataset.Dataset

Dataset Class for word-level model

Parameters

- **data_tensor** (*ins_num, seq_length*) – words
- **label_tensor** (*ins_num, seq_length*) – labels
- **mask_tensor** (*ins_num, seq_length*) – padding masks

class model.ner_dataset.**CRFDataset_WC**(*forw_tensor*, *forw_index*, *back_tensor*, *back_index*, *word_tensor*, *label_tensor*, *mask_tensor*, *len_tensor*)
Bases: torch.utils.data.dataset.Dataset

Dataset Class for char-aware model

Parameters

- **forw_tensor** (*ins_num, seq_length*) – forward chars
- **forw_index** (*ins_num, seq_length*) – index of forward chars
- **back_tensor** (*ins_num, seq_length*) – backward chars
- **back_index** (*ins_num, seq_length*) – index of backward chars
- **word_tensor** (*ins_num, seq_length*) – words
- **label_tensor** (*ins_num, seq_length*) – labels:

- **mask_tensor** (*ins_num, seq_length*) – padding masks
- **len_tensor** (*ins_num, 2*) – length of chars (dim0) and words (dim1)

1.8 model.utils module

model.utils.**adjust_learning_rate**(*optimizer*, *lr*)
shrink learning rate for pytorch

model.utils.**argmax**(*vec*)
helper function to calculate argmax of input vector at dimension 1

model.utils.**calc_threshold_mean**(*features*)
calculate the threshold for bucket by mean

model.utils.**concatChar**(*input_lines*, *char_dict*)
concat char into string

Parameters

- **input_lines** (*list of list of char*) – input corpus
- **char_dict** (*dictionary*) – char-level dictionary

Returns forw_lines

model.utils.**construct_bucket_gd**(*input_features*, *input_labels*, *thresholds*, *pad_feature*, *pad_label*)
Construct bucket by thresholds for greedy decode, word-level only

model.utils.**construct_bucket_mean_gd**(*input_features*, *input_label*, *word_dict*, *label_dict*)
Construct bucket by mean for greedy decode, word-level only

model.utils.**construct_bucket_mean_vb**(*input_features*, *input_label*, *word_dict*, *label_dict*, *caseless*)
Construct bucket by mean for viterbi decode, word-level only

model.utils.**construct_bucket_mean_vb_wc**(*word_features*, *input_label*, *label_dict*, *char_dict*, *word_dict*, *caseless*)
Construct bucket by mean for viterbi decode, word-level and char-level

model.utils.**construct_bucket_vb**(*input_features*, *input_labels*, *thresholds*, *pad_feature*, *pad_label*, *label_size*)
Construct bucket by thresholds for viterbi decode, word-level only

model.utils.**construct_bucket_vb_wc**(*word_features*, *forw_features*, *fea_len*, *input_labels*, *thresholds*, *pad_word_feature*, *pad_char_feature*, *pad_label*, *label_size*)
Construct bucket by thresholds for viterbi decode, word-level and char-level

model.utils.**encode**(*input_lines*, *word_dict*)
encode list of strings into word-level representation

model.utils.**encode2Tensor**(*input_lines*, *word_dict*, *unk*)
encode list of strings into word-level representation (tensor) with unk

model.utils.**encode2char_safe**(*input_lines*, *char_dict*)
get char representation of lines

Parameters

- **input_lines** (*list of strings*) – input corpus

- **char_dict** (*dictionary*) – char-level dictionary

Returns forw_lines

model.utils.**encode_corpus**(*lines*, *f_map*, *l_map*, *if_lower=False*)
encode corpus into features and labels

model.utils.**encode_corpus_c**(*lines*, *f_map*, *l_map*, *c_map*)
encode corpus into features (both word-level and char-level) and labels

model.utils.**encode_safe**(*input_lines*, *word_dict*, *unk*)
encode list of strings into word-level representation with unk

model.utils.**fill_y**(*nc*, *yidx*)
fill y to dense matrix

model.utils.**find_length_from_feats**(*feats*, *feat_to_ix*)
find length of unpadded features based on feature

model.utils.**find_length_from_labels**(*labels*, *label_to_ix*)
find length of unpadded features based on labels

model.utils.**generate_corpus**(*lines*, *if_shrink_feature=False*, *thresholds=1*)
generate label, feature, word dictionary and label dictionary

Parameters

- **lines** – corpus
- **if_shrink_feature** – whether shrink word-dictionary
- **threshold** – threshold for shrinking word-dictionary

model.utils.**generate_corpus_char**(*lines*, *if_shrink_c_feature=False*, *c_thresholds=1*, *if_shrink_w_feature=False*, *w_thresholds=1*)
generate label, feature, word dictionary, char dictionary and label dictionary

Parameters

- **lines** – corpus
- **if_shrink_c_feature** – whether shrink char-dictionary
- **c_threshold** – threshold for shrinking char-dictionary
- **if_shrink_w_feature** – whether shrink word-dictionary
- **w_threshold** – threshold for shrinking word-dictionary

model.utils.**init_embedding**(*input_embedding*)
Initialize embedding

model.utils.**init_linear**(*input_linear*)
Initialize linear transformation

model.utils.**init_lstm**(*input_lstm*)
Initialize lstm

model.utils.**iob_to_spans**(*sequence*, *lut*, *strict_iob2=False*)
convert to iob to span

model.utils.**iobes_to_spans**(*sequence*, *lut*, *strict_iob2=False*)
convert to iobes to span

model.utils.**load_embedding**(*emb_file*, *delimiter*, *feature_map*, *caseless*, *unk*, *shrink_to_train=False*)
load embedding

model.utils.**load_embedding_wlm**(*emb_file*, *delimiter*, *feature_map*, *full_feature_set*, *caseless*, *unk*, *emb_len*, *shrink_to_train=False*, *shrink_to_corpus=False*)
load embedding, indoc words would be listed before outdoc words

Parameters

- **emb_file** – path to embedding file
- **delimiter** – delimiter of lines
- **feature_map** – word dictionary
- **full_feature_set** – all words in the corpus
- **caseless** – convert into casesless style
- **unk** – string for unknown token
- **emb_len** – dimension of embedding vectors
- **shrink_to_train** – whether to shrink out-of-training set or not
- **shrink_to_corpus** – whether to shrink out-of-corpus or not

model.utils.**log_sum_exp**(*vec*, *m_size*)
calculate log of exp sum

Parameters

- **vec** (*batch_size, vanishing_dim, hidden_dim*) – input tensor
- **m_size** – hidden_dim

Returns batch_size, hidden_dim

model.utils.**read_corpus**(*lines*)
convert corpus into features and labels

model.utils.**read_features**(*lines*, *multi_docs=True*)
convert un-annotated corpus into features

model.utils.**revlut**(*lut*)

model.utils.**save_checkpoint**(*state*, *track_list*, *filename*)
save checkpoint

model.utils.**shrink_embedding**(*feature_map*, *word_dict*, *word_embedding*, *caseless*)
shrink embedding dictionary to in-doc words only

model.utils.**shrink_features**(*feature_map*, *features*, *thresholds*)
filter un-common features by threshold

model.utils.**switch**(*vec1*, *vec2*, *mask*)
switch function for pytorch

Parameters

- **vec1** (*any size*) – input tensor corresponding to 0
- **vec2** (*same to vec1*) – input tensor corresponding to 1
- **mask** (*same to vec1*) – input tensor, each element equals to 0/1

Returns vec (*)

model.utils.**to_scalar**(*var*)
change the first element of a tensor to scalar

1.9 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index

L

M

R

S

T

U

W