

---

# **LD-Net Documentation**

**Liyuan Liu**

**Dec 25, 2018**



<b>1</b>	<b>Language Modeling</b>	<b>3</b>
1.1	model_word_ada.adaptive module . . . . .	3
1.2	model_word_ada.basic module . . . . .	4
1.3	model_word_ada.dataset module . . . . .	5
1.4	model_word_ada.densenet module . . . . .	6
1.5	model_word_ada.lidnet module . . . . .	7
1.6	model_word_ada.LM module . . . . .	8
1.7	model_word_ada.utils module . . . . .	9
<b>2</b>	<b>Sequence Labeling</b>	<b>11</b>
2.1	model_seq.crf module . . . . .	11
2.2	model_seq.dataset module . . . . .	12
2.3	model_seq.elmo module . . . . .	13
2.4	model_seq.evaluator module . . . . .	15
2.5	model_seq.seqlabel module . . . . .	16
2.6	model_seq.seqlm module . . . . .	18
2.7	model_seq.sparse_lm module . . . . .	19
2.8	model_seq.utils module . . . . .	21
<b>3</b>	<b>Indices and tables</b>	<b>23</b>
	<b>Python Module Index</b>	<b>25</b>



**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 word-level language model, and sequence labeling with contextualized representation. The key feature of this project is the support of language model pruning without retraining.

Details about LD-Net can be accessed at: <https://arxiv.org/abs/1804.07827>.



## 1.1 model\_word\_ada.adaptive module

**class** model\_word\_ada.adaptive.**AdaptiveSoftmax** (*input\_size, cutoff*)

The adaptive softmax layer. Modified from: <https://github.com/rosinality/adaptive-softmax-pytorch/blob/master/adasoft.py>

**Parameters**

- **input\_size** (*int*, required.) – The input dimension.
- **cutoff** (*list*, required.) – The list of cutoff values.

**forward** (*w\_in, target*)

Calculate the log-likelihood w.o. calculate the full distribution.

**Parameters**

- **w\_in** (*torch.FloatTensor*, required.) – the input tensor, of shape (word\_num, input\_dim).
- **target** (*torch.FloatTensor*, required.) – the target of the language model, of shape (word\_num).

**Returns** **loss** – The NLL loss.

**Return type** *torch.FloatTensor*.

**log\_prob** (*w\_in, device*)

Calculate log-probability for the whole dictionary.

**Parameters**

- **w\_in** (*torch.FloatTensor*, required.) – the input tensor, of shape (word\_num, input\_dim).
- **device** (*torch.device*, required.) – the target device for calculation.

**Returns** **prob** – The full log-probability.

**Return type** `torch.FloatTensor`.

**`rand_ini()`**

Random Initialization.

## 1.2 `model_word_ada.basic` module

**`class model_word_ada.basic.BasicRNN`** (*layer\_num, unit, emb\_dim, hid\_dim, droprate*)

The multi-layer recurrent networks for the vanilla stacked RNNs.

### Parameters

- **`layer_num`** (`int`, required.) – The number of layers.
- **`unit`** (`torch.nn.Module`, required.) – The type of rnn unit.
- **`input_dim`** (`int`, required.) – The input dimension fo the unit.
- **`hid_dim`** (`int`, required.) – The hidden dimension fo the unit.
- **`droprate`** (`float`, required.) – The dropout ratrio.

**`forward(x)`**

Calculate the output.

**Parameters** **`x`** (`torch.LongTensor`, required.) – the input tensor, of shape (`seq_len`, `batch_size`, `input_dim`).

**Returns** **`output`** – The output of RNNs.

**Return type** `torch.FloatTensor`.

**`init_hidden()`**

Initialize hidden states.

**`rand_ini()`**

Random Initialization.

**`to_params()`**

To parameters.

**`class model_word_ada.basic.BasicUnit`** (*unit, input\_dim, hid\_dim, droprate*)

The basic recurrent unit for the vanilla stacked RNNs.

### Parameters

- **`unit`** (`str`, required.) – The type of rnn unit.
- **`input_dim`** (`int`, required.) – The input dimension fo the unit.
- **`hid_dim`** (`int`, required.) – The hidden dimension fo the unit.
- **`droprate`** (`float`, required.) – The dropout ratrio.

**`forward(x)`**

Calculate the output.

**Parameters** **`x`** (`torch.LongTensor`, required.) – the input tensor, of shape (`seq_len`, `batch_size`, `input_dim`).

**Returns** **`output`** – The output of RNNs.

**Return type** `torch.FloatTensor`.



**init\_hidden()**  
Initialize hidden states.

**rand\_ini()**  
Random Initialization.

### 1.3 model\_word\_ada.dataset module

**class** model\_word\_ada.dataset.**EvalDataset** (*dataset, sequence\_length*)  
Dataset for Language Modeling

**Parameters**

- **dataset** (*list*, required.) – The encoded dataset (outputs of preprocess scripts).
- **sequence\_length** (*int*, required.) – Sequence Length.

**construct\_index()**  
construct index for the dataset.

**get\_tqdm** (*device*)  
construct dataset reader and the corresponding tqdm.

**Parameters device** (*torch.device*, required.) – the target device for the dataset loader.

**reader** (*device*)  
construct dataset reader.

**Parameters device** (*torch.device*, required.) – the target device for the dataset loader.

**Returns reader** – A lazy iterable object

**Return type** *iterator*.

**class** model\_word\_ada.dataset.**LargeDataset** (*root, range\_idx, batch\_size, sequence\_length*)  
Lazy Dataset for Language Modeling

**Parameters**

- **root** (*str*, required.) – The root folder for dataset files.
- **range\_idx** (*int*, required.) – The maximum file index for the input files (train\_\*.pk).
- **batch\_size** (*int*, required.) – Batch size.
- **sequence\_length** (*int*, required.) – Sequence Length.

**get\_tqdm** (*device*)  
construct dataset reader and the corresponding tqdm.

**Parameters device** (*torch.device*, required.) – the target device for the dataset loader.

**open\_next()**  
Open the next file.

**reader** (*device*)  
construct dataset reader.

**Parameters device** (*torch.device*, required.) – the target device for the dataset loader.

**Returns reader** – A lazy iterable object

**Return type** *iterator*.

**shuffle()**  
shuffle dataset

## 1.4 model\_word\_ada.densenet module

**class** model\_word\_ada.densenet.**BasicUnit** (*unit, input\_dim, increase\_rate, droprate*)  
The basic recurrent unit for the densely connected RNNs.

### Parameters

- **unit** (`torch.nn.Module`, required.) – The type of rnn unit.
- **input\_dim** (`float`, required.) – The input dimension fo the unit.
- **increase\_rate** (`float`, required.) – The hidden dimension fo the unit.
- **droprate** (`float`, required.) – The dropout ratrio.

**forward** (*x*)  
Calculate the output.

**Parameters** **x** (`torch.LongTensor`, required.) – the input tensor, of shape (seq\_len, batch\_size, input\_dim).

**Returns** **output** – The output of RNNs.

**Return type** `torch.FloatTensor`.

**init\_hidden** ()  
Initialize hidden states.

**rand\_ini** ()  
Random Initialization.

**class** model\_word\_ada.densenet.**DenseRNN** (*layer\_num, unit, emb\_dim, hid\_dim, droprate*)  
The multi-layer recurrent networks for the densely connected RNNs.

### Parameters

- **layer\_num** (`float`, required.) – The number of layers.
- **unit** (`torch.nn.Module`, required.) – The type of rnn unit.
- **input\_dim** (`float`, required.) – The input dimension fo the unit.
- **hid\_dim** (`float`, required.) – The hidden dimension fo the unit.
- **droprate** (`float`, required.) – The dropout ratrio.

**forward** (*x*)  
Calculate the output.

**Parameters** **x** (`torch.LongTensor`, required.) – the input tensor, of shape (seq\_len, batch\_size, input\_dim).

**Returns** **output** – The output of RNNs.

**Return type** `torch.FloatTensor`.

**init\_hidden** ()  
Initialize hidden states.

**rand\_ini** ()  
Random Initialization.

**to\_params()**  
To parameters.

## 1.5 model\_word\_ada.ldnet module

**class** model\_word\_ada.ldnet.**BasicUnit** (*unit, input\_dim, increase\_rate, droprate, layer\_drop=0*)

The basic recurrent unit for the densely connected RNNs with layer-wise dropout.

### Parameters

- **unit** (`torch.nn.Module`, required.) – The type of rnn unit.
- **input\_dim** (`float`, required.) – The input dimension fo the unit.
- **increase\_rate** (`float`, required.) – The hidden dimension fo the unit.
- **droprate** (`float`, required.) – The dropout ratrio.
- **layer\_dropout** (`float`, required.) – The layer-wise dropout ratrio.

**forward** (*x, p\_out*)  
Calculate the output.

### Parameters

- **x** (`torch.LongTensor`, required.) – the input tensor, of shape (seq\_len, batch\_size, input\_dim).
- **p\_out** (`torch.LongTensor`, required.) – the final output tensor for the softmax, of shape (seq\_len, batch\_size, input\_dim).

### Returns

- **out** (`torch.FloatTensor`.) – The undropped outputs of RNNs to the softmax.
- **p\_out** (`torch.FloatTensor`.) – The dropped outputs of RNNs to the next\_layer.

**init\_hidden()**  
Initialize hidden states.

**rand\_ini()**  
Random Initialization.

**class** model\_word\_ada.ldnet.**LDRNN** (*layer\_num, unit, emb\_dim, hid\_dim, droprate, layer\_drop*)  
The multi-layer recurrent networks for the densely connected RNNs with layer-wise dropout.

### Parameters

- **layer\_num** (`float`, required.) – The number of layers.
- **unit** (`torch.nn.Module`, required.) – The type of rnn unit.
- **input\_dim** (`float`, required.) – The input dimension fo the unit.
- **hid\_dim** (`float`, required.) – The hidden dimension fo the unit.
- **droprate** (`float`, required.) – The dropout ratrio.
- **layer\_dropout** (`float`, required.) – The layer-wise dropout ratrio.

**forward** (*x*)  
Calculate the output.

**Parameters** **x** (`torch.LongTensor`, required.) – the input tensor, of shape (seq\_len, batch\_size, input\_dim).

**Returns** **output** – The output of RNNs to the Softmax.

**Return type** `torch.FloatTensor`.

**init\_hidden()**  
Initialize hidden states.

**rand\_ini()**  
Random Initialization.

**to\_params()**  
To parameters.

## 1.6 model\_word\_ada.LM module

**class** `model_word_ada.LM.LM`(*rnn*, *soft\_max*, *w\_num*, *w\_dim*, *droprate*, *label\_dim=-1*,  
*add\_relu=False*)

The language model model.

### Parameters

- **rnn** (`torch.nn.Module`, required.) – The RNNs network.
- **soft\_max** (`torch.nn.Module`, required.) – The softmax layer.
- **w\_num** (`int`, required.) – The number of words.
- **w\_dim** (`int`, required.) – The dimension of word embedding.
- **droprate** (`float`, required.) – The dropout ratio.
- **label\_dim** (`int`, required.) – The input dimension of softmax.

**forward** (*w\_in*, *target*)  
Calculate the loss.

### Parameters

- **w\_in** (`torch.FloatTensor`, required.) – the input tensor, of shape (word\_num, input\_dim).
- **target** (`torch.FloatTensor`, required.) – the target of the language model, of shape (word\_num).

**Returns** **loss** – The NLL loss.

**Return type** `torch.FloatTensor`.

**init\_hidden()**  
Initialize hidden states.

**load\_embed** (*origin\_lm*)  
Load embedding from another language model.

**log\_prob** (*w\_in*)  
Calculate log-probability for the whole dictionary.

**Parameters** **w\_in** (`torch.FloatTensor`, required.) – the input tensor, of shape (word\_num, input\_dim).

**Returns** **prob** – The full log-probability.

**Return type** `torch.FloatTensor`.

**rand\_ini()**

Random initialization.

## 1.7 model\_word\_ada.utils module

`model_word_ada.utils.adjust_learning_rate(optimizer, lr)`  
adjust learning to the the new value.

**Parameters**

- **optimizer** (*required.*) – pytorch optimizer.
- **float** (*float, required.*) – the target learning rate.

`model_word_ada.utils.init_embedding(input_embedding)`  
random initialize embedding

`model_word_ada.utils.init_linear(input_linear)`  
random initialize linear projection.

`model_word_ada.utils.init_lstm(input_lstm)`  
random initialize lstms

`model_word_ada.utils.repackage_hidden(h)`  
Wraps hidden states in new Variables, to detach them from their history

**Parameters** **h** (Tuple or Tensors, *required.*) – Tuple or Tensors, hidden states.

**Returns** **hidden** – detached hidden states

**Return type** Tuple or Tensors.

`model_word_ada.utils.to_scalar(var)`  
convert a tensor to a scalar number



## 2.1 model\_seq.crf module

**class** model\_seq.crf.CRF (*hidden\_dim: int, tagset\_size: int, if\_bias: bool = True*)  
Conditional Random Field Module

**Parameters**

- **hidden\_dim** (*int*, required.) – the dimension of the input features.
- **tagset\_size** (*int*, required.) – the size of the target labels.
- **if\_bias** (*bool*, optional, (default=True).) – whether the linear transformation has the bias term.

**forward** (*feats*)

calculate the potential score for the conditional random field.

**Parameters** **feats** (*torch.FloatTensor*, required.) – the input features for the conditional random field, of shape (\*, hidden\_dim).

**Returns** **output** – A float tensor of shape (ins\_num, from\_tag\_size, to\_tag\_size)

**Return type** *torch.FloatTensor*.

**rand\_init** ()

random initialization

**class** model\_seq.crf.CRFDecode (*y\_map: dict*)  
The negative loss for the Conditional Random Field Module

**Parameters** **y\_map** (*dict*, required.) – a dict maps from tag string to tag index.

**decode** (*scores, mask*)

find the best path from the potential scores by the viterbi decoding algorithm.

**Parameters**

- **scores** (`torch.FloatTensor`, required.) – the potential score for the conditional random field, of shape (`seq_len`, `batch_size`, `from_tag_size`, `to_tag_size`).
- **mask** (`torch.ByteTensor`, required.) – the mask for the unpadded sentence parts, of shape (`seq_len`, `batch_size`).

**Returns output** – A `LongTensor` of shape (`seq_len - 1`, `batch_size`)

**Return type** `torch.LongTensor`.

**to\_spans** (*sequence*)

decode the best path to spans.

**Parameters sequence** (*list*, *required.*) – the list of best label indexes paths .

**Returns output** – A set of chunks contains the position and type of the entities.

**Return type** `set`.

**class** `model_seq.crf.CRFLoss` (*y\_map: dict, average\_batch: bool = True*)

The negative loss for the Conditional Random Field Module

**Parameters**

- **y\_map** (`dict`, required.) – a `dict` maps from tag string to tag index.
- **average\_batch** (`bool`, optional, (default=`True`)). – whether the return score would be averaged per batch.

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

calculate the negative log likelihood for the conditional random field.

**Parameters**

- **scores** (`torch.FloatTensor`, required.) – the potential score for the conditional random field, of shape (`seq_len`, `batch_size`, `from_tag_size`, `to_tag_size`).
- **target** (`torch.LongTensor`, required.) – the positive path for the conditional random field, of shape (`seq_len`, `batch_size`).
- **mask** (`torch.ByteTensor`, required.) – the mask for the unpadded sentence parts, of shape (`seq_len`, `batch_size`).

**Returns loss** – The NLL loss.

**Return type** `torch.FloatTensor`.

## 2.2 model\_seq.dataset module

**class** `model_seq.dataset.SeqDataset` (*dataset: list, flm\_pad: int, blm\_pad: int, w\_pad: int, c\_con: int, c\_pad: int, y\_start: int, y\_pad: int, y\_size: int, batch\_size: int*)

Dataset for Sequence Labeling

**Parameters**

- **dataset** (`list`, required.) – The encoded dataset (outputs of preprocess scripts).
- **flm\_pad** (`int`, required.) – The pad index for the forward language model.
- **blm\_pad** (`int`, required.) – The pad index for the backward language model.
- **w\_pad** (`int`, required.) – The pad index for the word-level inputs.



- **c\_con** (*int*, required.) – The index of connect character token for character-level inputs.
- **c\_pad** (*int*, required.) – The pad index for the character-level inputs.
- **y\_start** (*int*, required.) – The index of the start label token.
- **y\_pad** (*int*, required.) – The index of the pad label token.
- **y\_size** (*int*, required.) – The size of the tag set.
- **batch\_size** (*int*, required.) – Batch size.

**batchify** (*batch*, *device*)

batchify a batch of data and move to a device.

**Parameters**

- **batch** (*list*, required.) – a sample from the encoded dataset (outputs of preprocess scripts).
- **device** (*torch.device*, required.) – the target device for the dataset loader.

**construct\_index** (*dataset*)

construct index for the dataset.

**Parameters dataset** (*list*, required.) – the encoded dataset (outputs of preprocess scripts).

**get\_tqdm** (*device*)

construct dataset reader and the corresponding tqdm.

**Parameters device** (*torch.device*, required.) – the target device for the dataset loader.

**reader** (*device*)

construct dataset reader.

**Parameters device** (*torch.device*, required.) – the target device for the dataset loader.

**Returns reader** – A lazy iterable object

**Return type** *iterator*.

**shuffle** ()

shuffle dataset

## 2.3 model\_seq.elmo module

**class** `model_seq.elmo.EBUnit` (*ori\_unit*, *droprate*, *fix\_rate*)

The basic recurrent unit for the ELMo RNNs wrapper.

**Parameters**

- **ori\_unit** (*torch.nn.Module*, required.) – The original module of rnn unit.
- **droprate** (*float*, required.) – The dropout ratio.
- **fix\_rate** (*bool*, required.) – Whether to fix the ratio.

**forward** (*x*)

Calculate the output.

**Parameters x** (*torch.FloatTensor*, required.) – The input tensor, of shape (seq\_len, batch\_size, input\_dim).

**Returns output** – The output of RNNs.

**Return type** `torch.FloatTensor`.

**class** `model_seq.elmo.ERNN` (*ori\_drnn, droprate, fix\_rate*)

The multi-layer recurrent networks for the ELMo RNNs wrapper.

**Parameters**

- **ori\_drnn** (`torch.nn.Module`, required.) – The original module of rnn networks.
- **droprate** (`float`, required.) – The dropout ratio.
- **fix\_rate** (`bool`, required.) – Whether to fix the ratio.

**forward** (*x*)

Calculate the output.

**Parameters** **x** (`torch.FloatTensor`, required.) – the input tensor, of shape (seq\_len, batch\_size, input\_dim).

**Returns** **output** – The ELMo outputs.

**Return type** `torch.FloatTensor`.

**regularizer** ()

Calculate the regularization term.

**Returns**

**Return type** The regularization term.

**class** `model_seq.elmo.ElmoLM` (*ori\_lm, backward, droprate, fix\_rate*)

The language model for the ELMo RNNs wrapper.

**Parameters**

- **ori\_lm** (`torch.nn.Module`, required.) – the original module of language model.
- **backward** (`bool`, required.) – whether the language model is backward.
- **droprate** (`float`, required.) – the dropout ratio.
- **fix\_rate** (`bool`, required.) – whether to fix the ratio.

**forward** (*w\_in, ind=None*)

Calculate the output.

**Parameters**

- **w\_in** (`torch.LongTensor`, required.) – the input tensor, of shape (seq\_len, batch\_size).
- **ind** (`torch.LongTensor`, optional, (default=None).) – the index tensor for the backward language model, of shape (seq\_len, batch\_size).

**Returns** **output** – The ELMo outputs.

**Return type** `torch.FloatTensor`.

**init\_hidden** ()

initialize hidden states.

**prox** (*lambda0*)

the proximal calculator.

**regularizer** ()

Calculate the regularization term.

**Returns** **reg** – The list of regularization terms.

Return type `list`.

## 2.4 model\_seq.evaluator module

**class** `model_seq.evaluator.eval_batch` (*decoder*)

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

**Parameters** **decoder** (`torch.nn.Module`, required.) – the decoder module, which needs to contain the `to_span()` method.

**acc\_score** ()

calculate the accuracy score based on the inner counter.

**calc\_acc\_batch** (*decoded\_data*, *target\_data*)

update statics for accuracy score.

**Parameters**

- **decoded\_data** (`torch.LongTensor`, required.) – the decoded best label index pathes.
- **target\_data** (`torch.LongTensor`, required.) – the golden label index pathes.

**calc\_f1\_batch** (*decoded\_data*, *target\_data*)

update statics for f1 score.

**Parameters**

- **decoded\_data** (`torch.LongTensor`, required.) – the decoded best label index pathes.
- **target\_data** (`torch.LongTensor`, required.) – the golden label index pathes.

**eval\_instance** (*best\_path*, *gold*)

Calculate statics to update inner counters for one instance.

**Parameters**

- **best\_path** (*required.*) – the decoded best label index pathe.
- **gold** (*required.*) – the golden label index pathes.

**f1\_score** ()

calculate the f1 score based on the inner counter.

**reset** ()

reset counters.

**class** `model_seq.evaluator.eval_wc` (*decoder*, *score\_type*)

evaluation class for LD-Net

**Parameters**

- **decoder** (`torch.nn.Module`, required.) – the decoder module, which needs to contain the `to_span()` and `decode()` method.
- **score\_type** (`str`, required.) – whether the f1 score or the accuracy is needed.

**calc\_score** (*seq\_model*, *dataset\_loader*)

calculate scores

**Parameters**

- **seq\_model** (*required.*) – sequence labeling model.

- **dataset\_loader** (*required.*) – the dataset loader.

**Returns** `score` – calculated score.

**Return type** `float`.

## 2.5 model\_seq.seqlabel module

```
class model_seq.seqlabel.SeqLabel (f_lm, b_lm, c_num: int, c_dim: int, c_hidden: int, c_layer:  
                                     int, w_num: int, w_dim: int, w_hidden: int, w_layer: int,  
                                     y_num: int, droprate: float, unit: str = 'lstm')
```

Sequence Labeling model augmented with language model.

### Parameters

- **f\_lm** (`torch.nn.Module`, *required.*) – The forward language model for contextualized representations.
- **b\_lm** (`torch.nn.Module`, *required.*) – The backward language model for contextualized representations.
- **c\_num** (`int`, *required.*) – The number of characters.
- **c\_dim** (`int`, *required.*) – The dimension of character embedding.
- **c\_hidden** (`int`, *required.*) – The dimension of character hidden states.
- **c\_layer** (`int`, *required.*) – The number of character lstms.
- **w\_num** (`int`, *required.*) – The number of words.
- **w\_dim** (`int`, *required.*) – The dimension of word embedding.
- **w\_hidden** (`int`, *required.*) – The dimension of word hidden states.
- **w\_layer** (`int`, *required.*) – The number of word lstms.
- **y\_num** (`int`, *required.*) – The number of tags types.
- **droprate** (`float`, *required.*) – The dropout ratio.
- **unit** (`"str"`, *optional*, (*default = 'lstm'*)) – The type of the recurrent unit.

```
forward (f_c, f_p, b_c, b_p, flm_w, blm_w, blm_ind, f_w)
```

Calculate the output (crf potentials).

### Parameters

- **f\_c** (`torch.LongTensor`, *required.*) – Character-level inputs in the forward direction.
- **f\_p** (`torch.LongTensor`, *required.*) – Output position of character-level inputs in the forward direction.
- **b\_c** (`torch.LongTensor`, *required.*) – Character-level inputs in the backward direction.
- **b\_p** (`torch.LongTensor`, *required.*) – Output position of character-level inputs in the backward direction.
- **flm\_w** (`torch.LongTensor`, *required.*) – Word-level inputs for the forward language model.

- **blm\_w** (`torch.LongTensor`, required.) – Word-level inputs for the backward language model.
- **blm\_ind** (`torch.LongTensor`, required.) – Output position of word-level inputs for the backward language model.
- **f\_w** (`torch.LongTensor`, required.) – Word-level inputs for the sequence labeling model.

**Returns output** – A float tensor of shape (sequence\_len, batch\_size, from\_tag\_size, to\_tag\_size)

**Return type** `torch.FloatTensor`.

**load\_pretrained\_word\_embedding** (*pre\_word\_embeddings*)

Load pre-trained word embedding.

**prune\_dense\_rnn** ()

Prune dense rnn to be smaller by deleting layers.

**rand\_init** ()

Random initialization.

**set\_batch\_seq\_size** (*sentence*)

Set the batch size and sequence length.

**to\_params** ()

To parameters.

**class** `model_seq.seqlabel.Vanilla_SeqLabel` (*f\_lm, b\_lm, c\_num, c\_dim, c\_hidden, c\_layer, w\_num, w\_dim, w\_hidden, w\_layer, y\_num, droprate, unit='lstm'*)

Sequence Labeling model augmented without language model.

#### Parameters

- **f\_lm** (`torch.nn.Module`, required.) – forward language module for contextualized representations.
- **b\_lm** (`torch.nn.Module`, required.) – backward language module for contextualized representations.
- **c\_num** (`int`, required.) – number of characters.
- **c\_dim** (`int`, required.) – dimension of character embedding.
- **c\_hidden** (`int`, required.) – dimension of character hidden states.
- **c\_layer** (`int`, required.) – number of character lstms.
- **w\_num** (`int`, required.) – number of words.
- **w\_dim** (`int`, required.) – dimension of word embedding.
- **w\_hidden** (`int`, required.) – dimension of word hidden states.
- **w\_layer** (`int`, required.) – number of word lstms.
- **y\_num** (`int`, required.) – number of tags types.
- **droprate** (`float`, required) – dropout ratio.
- **unit** ("*str*", optional, (default = '*lstm*')) – type of the recurrent unit.

**forward** (*f\_c, f\_p, b\_c, b\_p, flm\_w, blm\_w, blm\_ind, f\_w*)

Calculate the output (crf potentials).

**Parameters**

- **f\_c** (`torch.LongTensor`, required.) – Character-level inputs in the forward direction.
- **f\_p** (`torch.LongTensor`, required.) – Ouput position of character-level inputs in the forward direction.
- **b\_c** (`torch.LongTensor`, required.) – Character-level inputs in the backward direction.
- **b\_p** (`torch.LongTensor`, required.) – Ouput position of character-level inputs in the backward direction.
- **flm\_w** (`torch.LongTensor`, required.) – Word-level inputs for the forward language model.
- **blm\_w** (`torch.LongTensor`, required.) – Word-level inputs for the backward language model.
- **blm\_ind** (`torch.LongTensor`, required.) – Ouput position of word-level inputs for the backward language model.
- **f\_w** (`torch.LongTensor`, required.) – Word-level inputs for the sequence labeling model.

**Returns output** – A float tensor of shape (sequence\_len, batch\_size, from\_tag\_size, to\_tag\_size)

**Return type** `torch.FloatTensor`.

**load\_pretrained\_word\_embedding** (*pre\_word\_embeddings*)

Load pre-trained word embedding.

**rand\_init** ()

Random initialization.

**set\_batch\_seq\_size** (*sentence*)

set batch size and sequence length

## 2.6 model\_seq.seqlm module

**class** `model_seq.seqlm.BasicSeqLM` (*ori\_lm, backward, droprate, fix\_rate*)

The language model for the dense rnns.

**Parameters**

- **ori\_lm** (`torch.nn.Module`, required.) – the original module of language model.
- **backward** (`bool`, required.) – whether the language model is backward.
- **droprate** (`float`, required.) – the dropout ratrio.
- **fix\_rate** (`bool`, required.) – whether to fix the rqtio.

**forward** (*w\_in, ind=None*)

Calculate the output.

**Parameters**

- **w\_in** (`torch.LongTensor`, required.) – the input tensor, of shape (seq\_len, batch\_size).

- **ind** (`torch.LongTensor`, optional, (default=None).) – the index tensor for the backward language model, of shape (seq\_len, batch\_size).

**Returns** **output** – The ELMo outputs.

**Return type** `torch.FloatTensor`.

**init\_hidden**()  
initialize hidden states.

**regularizer**()  
Calculate the regularization term.

**Returns** **reg** – The list of regularization terms.

**Return type** `list`.

**to\_params**()  
To parameters.

## 2.7 model\_seq.sparse\_lm module

**class** `model_seq.sparse_lm.SBUnit` (*ori\_unit, droprate, fix\_rate*)  
The basic recurrent unit for the dense-RNNs wrapper.

### Parameters

- **ori\_unit** (`torch.nn.Module`, required.) – the original module of rnn unit.
- **droprate** (`float`, required.) – the dropout ratio.
- **fix\_rate** (`bool`, required.) – whether to fix the ratio.

**forward** (*x, weight=1*)  
Calculate the output.

### Parameters

- **x** (`torch.FloatTensor`, required.) – The input tensor, of shape (seq\_len, batch\_size, input\_dim).
- **weight** (`torch.FloatTensor`, required.) – The selection variable.

**Returns** **output** – The output of RNNs.

**Return type** `torch.FloatTensor`.

**prune\_rnn** (*mask*)  
Prune dense rnn to be smaller by selecting layers.

**Parameters** **mask** (`torch.ByteTensor`, required.) – The selection tensor for the input matrix.

**class** `model_seq.sparse_lm.SDRNN` (*ori\_drnn, droprate, fix\_rate*)  
The multi-layer recurrent networks for the dense-RNNs wrapper.

### Parameters

- **ori\_unit** (`torch.nn.Module`, required.) – the original module of rnn unit.
- **droprate** (`float`, required.) – the dropout ratio.
- **fix\_rate** (`bool`, required.) – whether to fix the ratio.

**forward** (*x*)

Calculate the output.

**Parameters** **x** (`torch.FloatTensor`, required.) – the input tensor, of shape (seq\_len, batch\_size, input\_dim).

**Returns** **output** – The ELMo outputs.

**Return type** `torch.FloatTensor`.

**prox** ()

the proximal calculator.

**prune\_dense\_rnn** ()

Prune dense rnn to be smaller by delecting layers.

**regularizer** ()

Calculate the regularization term.

**Returns**

- **reg0** (`torch.FloatTensor`.) – The value of reg0.
- **reg1** (`torch.FloatTensor`.) – The value of reg1.
- **reg2** (`torch.FloatTensor`.) – The value of reg2.

**to\_params** ()

To parameters.

**class** `model_seq.sparse_lm.SparseSeqLM` (*ori\_lm, backward, droprate, fix\_rate*)

The language model for the dense rnn with layer-wise selection.

**Parameters**

- **ori\_lm** (`torch.nn.Module`, required.) – the original module of language model.
- **backward** (`bool`, required.) – whether the language model is backward.
- **droprate** (`float`, required.) – the dropout ratrio.
- **fix\_rate** (`bool`, required.) – whether to fix the rqtio.

**forward** (*w\_in, ind=None*)

Calculate the output.

**Parameters**

- **w\_in** (`torch.LongTensor`, required.) – the input tensor, of shape (seq\_len, batch\_size).
- **ind** (`torch.LongTensor`, optional, (default=None).) – the index tensor for the backward language model, of shape (seq\_len, batch\_size).

**Returns** **output** – The ELMo outputs.

**Return type** `torch.FloatTensor`.

**init\_hidden** ()

initialize hidden states.

**prox** ()

the proximal calculator.

**prune\_dense\_rnn** ()

Prune dense rnn to be smaller by delecting layers.



**regularizer()**  
 Calculate the regularization term.

**Returns** **reg** – The list of regularization terms.

**Return type** `list`.

**to\_params()**  
 To parameters.

## 2.8 model\_seq.utils module

`model_seq.utils.adjust_learning_rate(optimizer, lr)`  
 adjust learning to the new value.

**Parameters**

- **optimizer** (*required.*) – pytorch optimizer.
- **float** (`float`, *required.*) – the target learning rate.

`model_seq.utils.init_embedding(input_embedding)`  
 random initialize embedding

`model_seq.utils.init_linear(input_linear)`  
 random initialize linear projection.

`model_seq.utils.init_lstm(input_lstm)`  
 random initialize lstms

`model_seq.utils.log_sum_exp(vec)`  
 log sum exp function.

**Parameters** **vec** (`torch.FloatTensor`, *required.*) – input vector, of shape(`ins_num`, `from_tag_size`, `to_tag_size`)

**Returns** **sum** – log sum exp results, tensor of shape (`ins_num`, `to_tag_size`)

**Return type** `torch.FloatTensor`.

`model_seq.utils.repackage_hidden(h)`  
 Wraps hidden states in new Variables, to detach them from their history

**Parameters** **h** (`Tuple` or `Tensors`, *required.*) – `Tuple` or `Tensors`, hidden states.

**Returns** **hidden** – detached hidden states

**Return type** `Tuple` or `Tensors`.

`model_seq.utils.to_scalar(var)`  
 convert a tensor to a scalar number



## CHAPTER 3

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### **a**

`adaptive`, 3

### **b**

`basic`, 4

### **c**

`crf`, 11

### **d**

`dataset`, 5

`densenet`, 6

### **e**

`elmo`, 13

`evaluator`, 15

### **l**

`ldnet`, 7

`LM`, 8

### **m**

`model_seq.crf`, 11

`model_seq.dataset`, 12

`model_seq.elmo`, 13

`model_seq.evaluator`, 15

`model_seq.seqlabel`, 16

`model_seq.seqlm`, 18

`model_seq.sparse_lm`, 19

`model_seq.utils`, 21

`model_word_ada.adaptive`, 3

`model_word_ada.basic`, 4

`model_word_ada.dataset`, 5

`model_word_ada.densenet`, 6

`model_word_ada.ldnet`, 7

`model_word_ada.LM`, 8

`model_word_ada.utils`, 9

`seqlm`, 18

`sparse_lm`, 19

### **u**

`utils`, 9

### **s**

`seqlabel`, 16



## A

acc\_score() (model\_seq.evaluator.eval\_batch method), 15  
 adaptive (module), 3  
 AdaptiveSoftmax (class in model\_word\_ada.adaptive), 3  
 adjust\_learning\_rate() (in module model\_seq.utils), 21  
 adjust\_learning\_rate() (in module model\_word\_ada.utils), 9

## B

basic (module), 4  
 BasicRNN (class in model\_word\_ada.basic), 4  
 BasicSeqLM (class in model\_seq.seqlm), 18  
 BasicUnit (class in model\_word\_ada.basic), 4  
 BasicUnit (class in model\_word\_ada.densenet), 6  
 BasicUnit (class in model\_word\_ada.ldnet), 7  
 batchify() (model\_seq.dataset.SeqDataset method), 13

## C

calc\_acc\_batch() (model\_seq.evaluator.eval\_batch method), 15  
 calc\_f1\_batch() (model\_seq.evaluator.eval\_batch method), 15  
 calc\_score() (model\_seq.evaluator.eval\_wc method), 15  
 construct\_index() (model\_seq.dataset.SeqDataset method), 13  
 construct\_index() (model\_word\_ada.dataset.EvalDataset method), 5  
 CRF (class in model\_seq.crf), 11  
 crf (module), 11  
 CRFDecode (class in model\_seq.crf), 11  
 CRFLoss (class in model\_seq.crf), 12

## D

dataset (module), 5, 12  
 decode() (model\_seq.crf.CRFDecode method), 11  
 densenet (module), 6  
 DenseRNN (class in model\_word\_ada.densenet), 6

## E

EUnit (class in model\_seq.elmo), 13

elmo (module), 13

ElmoLM (class in model\_seq.elmo), 14  
 ERNN (class in model\_seq.elmo), 14  
 eval\_batch (class in model\_seq.evaluator), 15  
 eval\_instance() (model\_seq.evaluator.eval\_batch method), 15  
 eval\_wc (class in model\_seq.evaluator), 15  
 EvalDataset (class in model\_word\_ada.dataset), 5  
 evaluator (module), 15

## F

f1\_score() (model\_seq.evaluator.eval\_batch method), 15  
 forward() (model\_seq.crf.CRF method), 11  
 forward() (model\_seq.crf.CRFLoss method), 12  
 forward() (model\_seq.elmo.EUnit method), 13  
 forward() (model\_seq.elmo.ElmoLM method), 14  
 forward() (model\_seq.elmo.ERNN method), 14  
 forward() (model\_seq.seqlabel.SeqLabel method), 16  
 forward() (model\_seq.seqlabel.Vanilla\_SeqLabel method), 17  
 forward() (model\_seq.seqlm.BasicSeqLM method), 18  
 forward() (model\_seq.sparse\_lm.SBUnit method), 19  
 forward() (model\_seq.sparse\_lm.SDRNN method), 19  
 forward() (model\_seq.sparse\_lm.SparseSeqLM method), 20  
 forward() (model\_word\_ada.adaptive.AdaptiveSoftmax method), 3  
 forward() (model\_word\_ada.basic.BasicRNN method), 4  
 forward() (model\_word\_ada.basic.BasicUnit method), 4  
 forward() (model\_word\_ada.densenet.BasicUnit method), 6  
 forward() (model\_word\_ada.densenet.DenseRNN method), 6  
 forward() (model\_word\_ada.ldnet.BasicUnit method), 7  
 forward() (model\_word\_ada.ldnet.LDRNN method), 7  
 forward() (model\_word\_ada.LM.LM method), 8

## G

get\_tqdm() (model\_seq.dataset.SeqDataset method), 13

`get_tqdm()` (model\_word\_ada.dataset.EvalDataset method), 5  
`get_tqdm()` (model\_word\_ada.dataset.LargeDataset method), 5

## I

`init_embedding()` (in module model\_seq.utils), 21  
`init_embedding()` (in module model\_word\_ada.utils), 9  
`init_hidden()` (model\_seq.elmo.ElmoLM method), 14  
`init_hidden()` (model\_seq.seqlm.BasicSeqLM method), 19  
`init_hidden()` (model\_seq.sparse\_lm.SparseSeqLM method), 20  
`init_hidden()` (model\_word\_ada.basic.BasicRNN method), 4  
`init_hidden()` (model\_word\_ada.basic.BasicUnit method), 4  
`init_hidden()` (model\_word\_ada.densenet.BasicUnit method), 6  
`init_hidden()` (model\_word\_ada.densenet.DenseRNN method), 6  
`init_hidden()` (model\_word\_ada.ldnet.BasicUnit method), 7  
`init_hidden()` (model\_word\_ada.ldnet.LDRNN method), 8  
`init_hidden()` (model\_word\_ada.LM.LM method), 8  
`init_linear()` (in module model\_seq.utils), 21  
`init_linear()` (in module model\_word\_ada.utils), 9  
`init_lstm()` (in module model\_seq.utils), 21  
`init_lstm()` (in module model\_word\_ada.utils), 9

## L

`LargeDataset` (class in model\_word\_ada.dataset), 5  
`ldnet` (module), 7  
`LDRNN` (class in model\_word\_ada.ldnet), 7  
`LM` (class in model\_word\_ada.LM), 8  
`LM` (module), 8  
`load_embed()` (model\_word\_ada.LM.LM method), 8  
`load_pretrained_word_embedding()` (model\_seq.seqlabel.SeqLabel method), 17  
`load_pretrained_word_embedding()` (model\_seq.seqlabel.Vanilla\_SeqLabel method), 18  
`log_prob()` (model\_word\_ada.adaptive.AdaptiveSoftmax method), 3  
`log_prob()` (model\_word\_ada.LM.LM method), 8  
`log_sum_exp()` (in module model\_seq.utils), 21

## M

`model_seq.crf` (module), 11  
`model_seq.dataset` (module), 12  
`model_seq.elmo` (module), 13  
`model_seq.evaluator` (module), 15

`model_seq.seqlabel` (module), 16  
`model_seq.seqlm` (module), 18  
`model_seq.sparse_lm` (module), 19  
`model_seq.utils` (module), 21  
`model_word_ada.adaptive` (module), 3  
`model_word_ada.basic` (module), 4  
`model_word_ada.dataset` (module), 5  
`model_word_ada.densenet` (module), 6  
`model_word_ada.ldnet` (module), 7  
`model_word_ada.LM` (module), 8  
`model_word_ada.utils` (module), 9

## O

`open_next()` (model\_word\_ada.dataset.LargeDataset method), 5

## P

`prox()` (model\_seq.elmo.ElmoLM method), 14  
`prox()` (model\_seq.sparse\_lm.SDRNN method), 20  
`prox()` (model\_seq.sparse\_lm.SparseSeqLM method), 20  
`prune_dense_rnn()` (model\_seq.seqlabel.SeqLabel method), 17  
`prune_dense_rnn()` (model\_seq.sparse\_lm.SDRNN method), 20  
`prune_dense_rnn()` (model\_seq.sparse\_lm.SparseSeqLM method), 20  
`prune_rnn()` (model\_seq.sparse\_lm.SBUnit method), 19

## R

`rand_ini()` (model\_word\_ada.adaptive.AdaptiveSoftmax method), 4  
`rand_ini()` (model\_word\_ada.basic.BasicRNN method), 4  
`rand_ini()` (model\_word\_ada.basic.BasicUnit method), 5  
`rand_ini()` (model\_word\_ada.densenet.BasicUnit method), 6  
`rand_ini()` (model\_word\_ada.densenet.DenseRNN method), 6  
`rand_ini()` (model\_word\_ada.ldnet.BasicUnit method), 7  
`rand_ini()` (model\_word\_ada.ldnet.LDRNN method), 8  
`rand_ini()` (model\_word\_ada.LM.LM method), 9  
`rand_init()` (model\_seq.crf.CRF method), 11  
`rand_init()` (model\_seq.seqlabel.SeqLabel method), 17  
`rand_init()` (model\_seq.seqlabel.Vanilla\_SeqLabel method), 18  
`reader()` (model\_seq.dataset.SeqDataset method), 13  
`reader()` (model\_word\_ada.dataset.EvalDataset method), 5  
`reader()` (model\_word\_ada.dataset.LargeDataset method), 5  
`regularizer()` (model\_seq.elmo.ElmoLM method), 14  
`regularizer()` (model\_seq.elmo.ERNN method), 14  
`regularizer()` (model\_seq.seqlm.BasicSeqLM method), 19  
`regularizer()` (model\_seq.sparse\_lm.SDRNN method), 20



regularizer() (model\_seq.sparse\_lm.SparseSeqLM method), 20  
 repackage\_hidden() (in module model\_seq.utils), 21  
 repackage\_hidden() (in module model\_word\_ada.utils), 9  
 reset() (model\_seq.evaluator.eval\_batch method), 15

## S

SBUnit (class in model\_seq.sparse\_lm), 19  
 SDRNN (class in model\_seq.sparse\_lm), 19  
 SeqDataset (class in model\_seq.dataset), 12  
 SeqLabel (class in model\_seq.seqlabel), 16  
 seqlabel (module), 16  
 seqlm (module), 18  
 set\_batch\_seq\_size() (model\_seq.seqlabel.SeqLabel method), 17  
 set\_batch\_seq\_size() (model\_seq.seqlabel.Vanilla\_SeqLabel method), 18  
 shuffle() (model\_seq.dataset.SeqDataset method), 13  
 shuffle() (model\_word\_ada.dataset.LargeDataset method), 5  
 sparse\_lm (module), 19  
 SparseSeqLM (class in model\_seq.sparse\_lm), 20

## T

to\_params() (model\_seq.seqlabel.SeqLabel method), 17  
 to\_params() (model\_seq.seqlm.BasicSeqLM method), 19  
 to\_params() (model\_seq.sparse\_lm.SDRNN method), 20  
 to\_params() (model\_seq.sparse\_lm.SparseSeqLM method), 21  
 to\_params() (model\_word\_ada.basic.BasicRNN method), 4  
 to\_params() (model\_word\_ada.densenet.DenseRNN method), 6  
 to\_params() (model\_word\_ada.ldnet.LDRNN method), 8  
 to\_scalar() (in module model\_seq.utils), 21  
 to\_scalar() (in module model\_word\_ada.utils), 9  
 to\_spans() (model\_seq.crf.CRFDecode method), 12

## U

utils (module), 9, 21

## V

Vanilla\_SeqLabel (class in model\_seq.seqlabel), 17