Emmental Documentation

Sen Wu

Feb 04, 2022

USER DOCUMENTATION

Emmental is a framework for building multi-modal multi-task deep learning systems.

Note that Emmental is still *actively under development*, so feedback and contributions are welcome. Submit bugs in the Issues section or feel free to submit your contributions as a pull request.

CHAPTER

ONE

GETTING STARTED

This document will show you how to get up and running with Emmental. We'll show you how to get everything installed on your machine so that you can walk through real examples by checking out our Tutorials.

1.1 Installing the Emmental Package

Install Emmental by running:

```
$ pip install emmental
```

Note: Emmental only supports Python 3. Python 2 is not supported.

Tip: For the Python dependencies, we recommend using a virtualenv, which will allow you to install Emmental and its python dependencies in an isolated Python environment. Once you have virtualenv installed, you can create a Python 3 virtual environment as follows.:

```
$ virtualenv -p python3.6 .venv
```

Once the virtual environment is created, activate it by running:

```
$ source .venv/bin/activate
```

Any Python libraries installed will now be contained within this virtual environment. To deactivate the environment, simply run:

```
$ deactivate
```

1.2 The Emmental Framework

The Emmental framework can be broken into four components.

1. Dataset and Dataloader

   In this first component, the users' input is parsed into Emmental's dataset and then feed into Emmental's dataloader.

2. Task

   In this component, we let user to use declarative language-like way to defeine the taksk, which includes task name (name), module used in the task (module_pool), task flow (task_flow), loss function used in the task (loss_func), output function (output_func), and the score functions (scorer).

3. Model

   Here, we initialize the Emmental model with the Emmental tasks. Users can define different types of models, such as single-task model, multi-task model, multi-modality task.

4. Learning

   Finally, Emmental provides learning component which is used to train the Emmental model. Optionally, users can use different training schedulers during learning process.

To demonstrate how to set up and use Emmental in your applications, we walk through each of these phases in real-world examples in our Tutorials.

CHAPTER

TWO

DATASET AND DATALOADER

The first component of Emmental's pipeline is to use user provided data to create Emmental Dataset and Dataloader.

2.1 Dataset and Dataloader Classes

The following docs describe elements of Emmental's Dataset and Dataloader.

Emmental dataset and dataloader.

class emmental.data.**EmmentalDataLoader**(*task_to_label_dict*, *dataset*, *split='train'*, *collate_fn=None*, *n_batches=None*, ***kwargs*)

Bases: torch.utils.data.dataloader.DataLoader

Emmental dataLoader.

An advanced dataloader class which contains mapping from task to label (which label(s) to use in dataset's Y_dict for this task), and split (which part this dataset belongs to) information.

Parameters

- **task_to_label_dict** (Dict[str, str]) – The task to label mapping where key is the task name and value is the label(s) for that task and should be the key in Y_dict.
- **dataset** (*EmmentalDataset*) – The dataset to construct the dataloader
- **split** (str) – The split information, defaults to "train".
- **collate_fn** (Optional[Callable]) – The function that merges a list of samples to form a mini-batch, defaults to emmental_collate_fn.
- **n_batches** (Optional[int]) – Total number of batches.
- ****Kwargs** – Other arguments of dataloader.

class emmental.data.**EmmentalDataset**(*name*, *X_dict*, *Y_dict=None*, *uid=None*)

Bases: torch.utils.data.dataset.Dataset

Emmental dataset.

An advanced dataset class to handle that the input data contains multiple fields and the output data contains multiple label sets.

Parameters

- **name** (str) – The name of the dataset.
- **X_dict** (Dict[str, Any]) – The feature dict where key is the feature name and value is the feature.

- **Y_dict** (Optional[Dict[str, Tensor]]) – The label dict where key is the label name and value is the label, defaults to None.
- **uid** (Optional[str]) – The unique id key in the X_dict, defaults to None.

add_features(*X_dict*)
Add new features into X_dict.

Parameters **X_dict** (Dict[str, Any]) – The new feature dict to add into the existing feature dict.

Return type None

add_labels(*Y_dict*)
Add new labels into Y_dict.

Parameters **Y_dict** (Dict[str, Tensor]) – the new label dict to add into the existing label dict

Return type None

remove_feature(*feature_name*)
Remove one feature from feature dict.

Parameters **feature_name** (str) – The feature that removes from feature dict.

Return type None

remove_label(*label_name*)
Remove one label from label dict.

Parameters **label_name** (str) – The label that removes from label dict.

Return type None

emmental.data.**emmental_collate_fn**(*batch*, *min_data_len=0*, *max_data_len=0*)
Collate function.

Parameters

- **batch** (Union[List[Tuple[Dict[str, Any], Dict[str, Tensor]]], List[Dict[str, Any]]]) – The batch to collate.
- **min_data_len** (int) – The minimal data sequence length, defaults to 0.
- **max_data_len** (int) – The maximal data sequence length (0 means no limit), defaults to 0.

Return type Union[Tuple[Dict[str, Any], Dict[str, Tensor]], Dict[str, Any]]

Returns The collated batch.

2.2 Configuration Settings

Visit the Configuring Emmental page to see how to provide configuration parameters to Emmental via `.emmental-config.yaml`.

The parameters of data are described below:

```
# Data configuration
data_config:
    min_data_len: 0 # min data length
    max_data_len: 0 # max data length (e.g., 0 for no max_len)
```

CHAPTER

THREE

TASK

The second component of Emmental's pipeline is to build learning Task.

3.1 Task Class

The following describes elements of used for creating Task.

Emmental task.

class emmental.task.**Action**(*name*, *module*, *inputs=None*)

Bases: object

An action to execute in a EmmentalTask task_flow.

Action is the object that populate the task_flow sequence. It has three attributes: name, module_name and inputs where name is the name of the action, module_name is the module name used in this action and inputs is the inputs to the action. By introducing a class for specifying actions in the task_flow, we standardize its definition. Moreover, Action enables more user flexibility in specifying a task flow as we can now support a wider-range of formats for the input attribute of a task_flow as follow:

1. It now supports str as inputs (e.g., inputs="input1") which means take the input1's output as input for current action.

2. It also support None as inputs which will take all modules' output as input.

3. It also supports a list as inputs which can be constructed by three different formats:

a). x (x is str) where takes whole output of x's output as input: this enables users to pass all outputs from one module to another without having to manually specify every input to the module.

b). (x, y) (y is int) where takes x's y-th output as input.

c). (x, y) (y is str) where takes x's output str as input.

Parameters

- **name** (str) – The name of the action.
- **module_name** – The module_name of the module.
- **inputs** (Union[str, Sequence[Union[str, Tuple[str, str], Tuple[str, int]]], None]) – The inputs of the action. Details can be found above.

class emmental.task.**EmmentalTask**(*name*, *module_pool*, *task_flow*, *loss_func*, *output_func*, *scorer=None*, *action_outputs=None*, *module_device={}*, *weight=1.0*, *require_prob_for_eval=True*, *require_pred_for_eval=True*)

Bases: object

Task class to define task in Emmental model.

Parameters

- **name** (str) – The name of the task (Primary key).
- **module_pool** (ModuleDict) – A dict of modules that uses in the task.
- **task_flow** (Sequence[*Action*]) – The task flow among modules to define how the data flows.
- **loss_func** (Callable) – The function to calculate the loss.
- **output_func** (Callable) – The function to generate the output.
- **scorer** (Optional[*Scorer*]) – The class of metrics to evaluate the task, defaults to None.
- **action_outputs** (Union[str, Sequence[Union[str, Tuple[str, str], Tuple[str, int]]], None]) – The action outputs need to output, defaults to None.
- **module_device** (Dict[str, Union[int, str, device]]) – The dict of module device specification, defaults to None.
- **weight** (Union[float, int]) – The weight of the task, defaults to 1.0.
- **require_prob_for_eval** (bool) – Whether require prob for evaluation, defaults to True.
- **require_pred_for_eval** (bool) – Whether require pred for evaluation, defaults to True.

3.2 Task Utilities

These utilities are used to build task.

Emmental scorer.

class emmental.scorer.**Scorer**(*metrics=[]*, *customize_metric_funcs={}*)

Bases: object

A class to score tasks.

Parameters

- **metrics** (List[str]) – A list of metric names which provides in emmental (e.g., accuracy), defaults to [].
- **customize_metric_funcs** (Dict[str, Callable]) – a dict of customize metric where key is the metric name and value is the metric function which takes golds, preds, probs, uids as input, defaults to {}.

score(*golds*, *preds*, *probs*, *uids=None*)

Calculate the score.

Parameters

- **golds** (Union[ndarray, List[ndarray]]) – Ground truth values.
- **probs** (Union[ndarray, List[ndarray]]) – Predicted probabilities.
- **preds** (Union[ndarray, List[ndarray]]) – Predicted values.
- **uids** (Optional[List[str]]) – Unique ids, defaults to None.

Return type Dict[str, float]

Returns Score dict.

3.3 Metrics

This show the metrics included with Emmental. These metrics can be used alone, or combined together, to define how to evaluate the task.

Emmental metric module.

emmental.metrics.**accuracy_f1_scorer**(*golds*, *probs*, *preds*, *uids=None*, *pos_label=1*)
Average of accuracy and f1 score.

Parameters

- **golds** (ndarray) – Ground truth values.
- **probs** (Optional[ndarray]) – Predicted probabilities.
- **preds** (ndarray) – Predicted values.
- **uids** (Optional[List[str]]) – Unique ids, defaults to None.
- **pos_label** (int) – The positive class label, defaults to 1.

Return type Dict[str, float]

Returns Average of accuracy and f1.

emmental.metrics.**accuracy_scorer**(*golds*, *probs*, *preds*, *uids=None*, *normalize=True*, *topk=1*)
Accuracy classification score.

Parameters

- **golds** (ndarray) – Ground truth values.
- **probs** (Optional[ndarray]) – Predicted probabilities.
- **preds** (Optional[ndarray]) – Predicted values.
- **uids** (Optional[List[str]]) – Unique ids, defaults to None.
- **normalize** (bool) – Normalize the results or not, defaults to True.
- **topk** (int) – Top K accuracy, defaults to 1.

Return type Dict[str, Union[float, int]]

Returns Accuracy, if normalize is True, return the fraction of correctly predicted samples (float), else returns the number of correctly predicted samples (int).

emmental.metrics.**f1_scorer**(*golds*, *probs*, *preds*, *uids=None*, *pos_label=1*)
F-1 score.

Parameters

- **golds** (ndarray) – Ground truth values.
- **probs** (Optional[ndarray]) – Predicted probabilities.
- **preds** (ndarray) – Predicted values.
- **uids** (Optional[List[str]]) – Unique ids.
- **pos_label** (int) – The positive class label, defaults to 1.

Return type Dict[str, float]

Returns F-1 score.

emmental.metrics.**fbeta_scorer**(*golds*, *probs*, *preds*, *uids=None*, *pos_label=1*, *beta=1*)
F-beta score is the weighted harmonic mean of precision and recall.

Parameters

- **golds** (ndarray) – Ground truth values.
- **probs** (Optional[ndarray]) – Predicted probabilities.
- **preds** (ndarray) – Predicted values.
- **uids** (Optional[List[str]]) – Unique ids, defaults to None.
- **pos_label** (int) – The positive class label, defaults to 1.
- **beta** (int) – Weight of precision in harmonic mean, defaults to 1.

Return type Dict[str, float]

Returns F-beta score.

emmental.metrics.**matthews_correlation_coefficient_scorer**(*golds*, *probs*, *preds*, *uids=None*)
Matthews correlation coefficient (MCC).

Parameters

- **golds** (ndarray) – Ground truth values.
- **probs** (Optional[ndarray]) – Predicted probabilities.
- **preds** (ndarray) – Predicted values.
- **uids** (Optional[List[str]]) – Unique ids, defaults to None.

Return type Dict[str, float]

Returns Matthews correlation coefficient score.

emmental.metrics.**mean_squared_error_scorer**(*golds*, *probs*, *preds*, *uids=None*)
Mean squared error regression loss.

Parameters

- **golds** (ndarray) – Ground truth values.
- **probs** (ndarray) – Predicted probabilities.
- **preds** (Optional[ndarray]) – Predicted values.
- **uids** (Optional[List[str]]) – Unique ids, defaults to None.

Return type Dict[str, float]

Returns Mean squared error regression loss.

emmental.metrics.**pearson_correlation_scorer**(*golds*, *probs*, *preds*, *uids=None*, *return_pvalue=False*)
Pearson correlation coefficient and the p-value.

Parameters

- **golds** (ndarray) – Ground truth values.
- **probs** (ndarray) – Predicted probabilities.
- **preds** (Optional[ndarray]) – Predicted values.
- **uids** (Optional[List[str]]) – Unique ids, defaults to None.
- **return_pvalue** (bool) – Whether return pvalue or not, defaults to False.

Return type Dict[str, float]

Returns Pearson correlation coefficient with pvalue if return_pvalue is True.

emmental.metrics.**pearson_spearman_scorer**(*golds*, *probs*, *preds*, *uids=None*)
Average of Pearson and Spearman rank-order correlation coefficients.

Parameters

- **golds** (ndarray) – Ground truth values.
- **probs** (ndarray) – Predicted probabilities.
- **preds** (Optional[ndarray]) – Predicted values.
- **uids** (Optional[List[str]]) – Unique ids, defaults to None.

Return type Dict[str, float]

Returns The average of Pearson correlation coefficient and Spearman rank-order correlation coefficient.

emmental.metrics.**precision_scorer**(*golds*, *probs*, *preds*, *uids=None*, *pos_label=1*)
Precision.

Parameters

- **golds** (ndarray) – Ground truth values.
- **probs** (Optional[ndarray]) – Predicted probabilities.
- **preds** (ndarray) – Predicted values.
- **uids** (Optional[List[str]]) – Unique ids, defaults to None.
- **pos_label** (int) – The positive class label, defaults to 1.

Return type Dict[str, float]

Returns Precision.

emmental.metrics.**recall_scorer**(*golds*, *probs*, *preds*, *uids=None*, *pos_label=1*)
Recall.

Parameters

- **golds** (ndarray) – Ground truth values.
- **probs** (Optional[ndarray]) – Predicted probabilities.
- **preds** (ndarray) – Predicted values.
- **uids** (Optional[List[str]]) – Unique ids, defaults to None.
- **pos_label** (int) – The positive class label, defaults to 1.

Return type Dict[str, float]

Returns Recall.

emmental.metrics.**roc_auc_scorer**(*golds*, *probs*, *preds*, *uids=None*)
ROC AUC.

Parameters

- **golds** (ndarray) – Ground truth values.
- **probs** (ndarray) – Predicted probabilities.
- **preds** (Optional[ndarray]) – Predicted values.

- **uids** (Optional[List[str]]) – Unique ids, defaults to None.
- **pos_label** – The positive class label, defaults to 1.

Return type Dict[str, float]

Returns ROC AUC score.

emmental.metrics.**spearman_correlation_scorer**(*golds*, *probs*, *preds*, *uids=None*, *return_pvalue=False*)

Spearman rank-order correlation coefficient and the p-value.

Parameters

- **golds** (ndarray) – Ground truth values.
- **probs** (ndarray) – Predicted probabilities.
- **preds** (Optional[ndarray]) – Predicted values.
- **uids** (Optional[List[str]]) – Unique ids, defaults to None.
- **return_pvalue** (bool) – Whether return pvalue or not, defaults to False.

Return type Dict[str, float]

Returns Spearman rank-order correlation coefficient with pvalue if return_pvalue is True.

CHAPTER

FOUR

MODEL

The third component of Emmental's pipeline is to build deep learning model with your tasks.

4.1 Emmental Model

The following describes elements of used for model creation.

Emmental model.

class emmental.model.**EmmentalModel**(*name=None*, *tasks=None*)

Bases: torch.nn.modules.module.Module

A class to build multi-task model.

Parameters

- **name** (Optional[str]) – Name of the model, defaults to None.
- **tasks** (Union[*EmmentalTask*, List[*EmmentalTask*], None]) – A task or a list of tasks.

add_task(*task*)

Add a single task into MTL network.

Parameters **task** (*EmmentalTask*) – A task to add.

Return type None

add_tasks(*tasks*)

Build the MTL network using all tasks.

Parameters **tasks** (Union[*EmmentalTask*, List[*EmmentalTask*]]) – A task or a list of tasks.

Return type None

collect_state_dict()

Collect the state dict.

Return type Dict[str, Any]

flow(*X_dict*, *task_names*)

Forward based on input and task flow.

Note: We assume that all shared modules from all tasks are based on the same input.

Parameters

- **X_dict** (Dict[str, Any]) – The input data

- **task_names** (List[str]) – The task names that needs to forward.

Return type Dict[str, Any]

Returns The output of all forwarded modules

forward(*uids*, *X_dict*, *Y_dict*, *task_to_label_dict*, *return_loss=True*, *return_probs=True*, *return_action_outputs=False*)

Forward function.

Parameters

- **uids** (List[str]) – The uids of input data.
- **X_dict** (Dict[str, Any]) – The input data.
- **Y_dict** (Dict[str, Tensor]) – The output data.
- **task_to_label_dict** (Dict[str, str]) – The task to label mapping.
- **return_loss** – Whether return loss or not, defaults to True.
- **return_probs** – Whether return probs or not, defaults to True.
- **return_action_outputs** – Whether return action_outputs or not,
- **False.** (*defaults to*) –

Return type Union[Tuple[Dict[str, List[str]], Dict[str, Tensor], Dict[str, Union[ndarray, List[ndarray]]], Dict[str, Union[ndarray, List[ndarray]]], Dict[str, Dict[str, Union[ndarray, List]]]], Tuple[Dict[str, List[str]], Dict[str, Tensor], Dict[str, Union[ndarray, List[ndarray]]], Dict[str, Union[ndarray, List[ndarray]]]]]

Returns The uids, loss, prob, gold, action_output (optional) in the batch of all tasks.

load(*model_path*, *verbose=True*)

Load model state_dict from file and reinitialize the model weights.

Parameters

- **model_path** (str) – Saved model path.
- **verbose** (bool) – Whether log the info, defaults to *True*.

Return type None

load_state_dict(*state_dict*)

Load the state dict.

Parameters **state_dict** (Dict[str, Any]) – The state dict to load.

Return type None

predict(*dataloader*, *return_loss=True*, *return_probs=True*, *return_preds=False*, *return_action_outputs=False*)

Predict from dataloader.

Parameters

- **dataloader** (*EmmentalDataLoader*) – The dataloader to predict.
- **return_loss** (bool) – Whether return loss or not, defaults to True.
- **return_probs** (bool) – Whether return probs or not, defaults to True.
- **return_preds** (bool) – Whether return predictions or not, defaults to False.

- **return_action_outputs** (bool) – Whether return action_outputs or not,
- **False.** (*defaults to*) –

Return type Dict[str, Any]

Returns The result dict.

remove_task(*task_name*)
Remove a existing task from MTL network.

Parameters **task_name** (str) – The task name to remove.

Return type None

save(*model_path*, *iteration=None*, *metric_dict=None*, *verbose=True*)
Save model.

Parameters

- **model_path** (str) – Saved model path.
- **iteration** (Union[float, int, None]) – The iteration of the model, defaults to *None*.
- **metric_dict** (Optional[Dict[str, float]]) – The metric dict, defaults to *None*.
- **verbose** (bool) – Whether log the info, defaults to *True*.

Return type None

score(*dataloaders*, *return_average=True*)
Score the data from dataloader.

Parameters

- **dataloaders** (Union[*EmmentalDataLoader*, List[*EmmentalDataLoader*]]) – The dataloaders to score.
- **return_average** (bool) – Whether to return average score.

Return type Dict[str, float]

Returns Score dict.

update_task(*task*)
Update a existing task in MTL network.

Parameters **task** (*EmmentalTask*) – A task to update.

Return type None

4.2 Configuration Settings

Visit the Configuring Emmental page to see how to provide configuration parameters to Emmental via `.emmental-config.yaml`.

The model parameters are described below:

```
# Model configuration
model_config:
    model_path: # path to pretrained model
    device: 0 # -1 for cpu or gpu id (e.g., 0 for cuda:0)
```

(continues on next page)

(continued from previous page)

```
    dataparallel: True # whether to use dataparallel or not
    distributed_backend: nccl # what distributed backend to use for DDP [nccl, gloo]
```

CHAPTER

FIVE

LEARNING

The final component of Emmental's pipeline is to learn the user defined deep learning model based user defined data.

5.1 Core Learning Objects

These are Emmental's core objects used for learning.

Emmental learner.

class emmental.learner.**EmmentalLearner**(*name=None*)

Bases: object

A class for emmental multi-task learning.

Parameters **name** (Optional[str]) – Name of the learner, defaults to None.

learn(*model*, *dataloaders*)

Learning procedure of emmental MTL.

Parameters

- **model** (*EmmentalModel*) – The emmental model that needs to learn.
- **dataloaders** (List[*EmmentalDataLoader*]) – A list of dataloaders used to learn the model.

Return type None

5.2 Schedulers

These are several schedulers supported in Emmental learner.

Emmental scheduler module.

class emmental.schedulers.**MixedScheduler**(*fillup=False*)

Bases: emmental.schedulers.scheduler.Scheduler

Generate batch generator from all dataloaders in mixture for MTL training.

Parameters **fillup** (bool) – Whether fillup to make all dataloader the same size.

get_batches(*dataloaders*, *model=None*)

Generate batch generator from all dataloaders in mixture for one epoch.

Parameters

- **dataloaders** (List[*EmmentalDataLoader*]) – List of dataloaders.

- **model** (Optional[*EmmentalModel*]) – The training model, defaults to None.

Return type Iterator[Union[Batch, List[Batch]]]

Returns A generator of all batches.

get_num_batches(*dataloaders*)
Get total number of batches per epoch.

Parameters **dataloaders** (List[*EmmentalDataLoader*]) – List of dataloaders.

Return type int

Returns Total number of batches per epoch.

class emmental.schedulers.**RoundRobinScheduler**(*fillup=False*)
Bases: emmental.schedulers.scheduler.Scheduler

Generate batch generator from all dataloaders in round robin order.

Parameters **fillup** (bool) – Whether fillup to make all dataloader the same size.

get_batches(*dataloaders*, *model=None*)
Generate batch generator from all dataloaders for one epoch.

Parameters

- **dataloaders** (List[*EmmentalDataLoader*]) – List of dataloaders.
- **model** (Optional[*EmmentalModel*]) – The training model, defaults to None.

Return type Iterator[Union[Batch, List[Batch]]]

Returns A generator of all batches.

get_num_batches(*dataloaders*)
Get total number of batches per epoch.

Parameters **dataloaders** (List[*EmmentalDataLoader*]) – List of dataloaders.

Return type int

Returns Total number of batches per epoch.

class emmental.schedulers.**SequentialScheduler**(*fillup=False*)
Bases: emmental.schedulers.scheduler.Scheduler

Generate batch generator from all dataloaders in sequential order.

Parameters **fillup** (bool) – Whether fillup to make all dataloader the same size.

get_batches(*dataloaders*, *model=None*)
Generate batch generator from all dataloaders for one epoch.

Parameters

- **dataloaders** (List[*EmmentalDataLoader*]) – List of dataloaders.
- **model** (Optional[*EmmentalModel*]) – The training model, defaults to None.

Return type Iterator[Union[Batch, List[Batch]]]

Returns A generator of all batches.

get_num_batches(*dataloaders*)
Get total number of batches per epoch.

Parameters **dataloaders** (List[*EmmentalDataLoader*]) – List of dataloaders.

Return type int

Returns Total number of batches per epoch.

5.3 Configuration Settings

Visit the Configuring Emmental page to see how to provide configuration parameters to Emmental via `.emmental-config.yaml`.

The learning parameters of the model are described below:

```
# Learning configuration
learner_config:
    optimizer_path: # path to optimizer state
    scheduler_path: # path to lr scheduler state
    fp16: False # whether to use 16-bit precision
    fp16_opt_level: O1 # Apex AMP optimization level (e.g., ['O0', 'O1', 'O2', 'O3'])
    local_rank: -1 # local_rank for distributed training on gpus
    epochs_learned: 0 # learning epochs learned
    n_epochs: 1 # total number of learning epochs
    steps_learned: 0 # learning steps learned
    n_steps: # total number of learning steps
    skip_learned_data: False # skip learned batches if steps_learned or epochs_learned␣
↪nonzero
    train_split: # the split for training, accepts str or list of strs
        - train
    valid_split: # the split for validation, accepts str or list of strs
        - valid
    test_split: # the split for testing, accepts str or list of strs
        - test
    online_eval: 0 # whether to perform online evaluation
    optimizer_config:
        optimizer: adam # [sgd, adam, adamax, bert_adam]
        parameters: # parameters to optimize
        lr: 0.001 # Learing rate
        l2: 0.0 # l2 regularization
        grad_clip: # gradient clipping
        gradient_accumulation_steps: 1 # gradient accumulation steps
        asgd_config:
            lambd: 0.0001
            alpha: 0.75
            t0: 1000000.0
        adadelta_config:
            rho: 0.9
            eps: 0.000001
        adagrad_config:
            lr_decay: 0
            initial_accumulator_value: 0
            eps: 0.0000000001
        adam_config:
            betas: !!python/tuple [0.9, 0.999]
            eps: 0.00000001
            amsgrad: False
```

(continues on next page)

(continued from previous page)

```
        adamw_config:
            betas: !!python/tuple [0.9, 0.999]
            eps: 0.00000001
            amsgrad: False
        adamax_config:
            betas: !!python/tuple [0.9, 0.999]
            eps: 0.00000001
        lbfgs_config:
            max_iter: 20
            max_eval:
            tolerance_grad: 0.0000001
            tolerance_change: 0.000000001
            history_size: 100
            line_search_fn:
        rms_prop_config:
            alpha: 0.99
            eps: 0.00000001
            momentum: 0
            centered: False
        r_prop_config:
            etas: !!python/tuple [0.5, 1.2]
            step_sizes: !!python/tuple [0.000001, 50]
        sgd_config:
            momentum: 0
            dampening: 0
            nesterov: False
        sparse_adam_config:
            betas: !!python/tuple [0.9, 0.999]
            eps: 0.00000001
        bert_adam_config:
            betas: !!python/tuple [0.9, 0.999]
            eps: 0.00000001
    lr_scheduler_config:
        lr_scheduler: # [linear, exponential, reduce_on_plateau, cosine_annealing]
        lr_scheduler_step_unit: batch # [batch, epoch]
        lr_scheduler_step_freq: 1
        warmup_steps: # warm up steps
        warmup_unit: batch # [epoch, batch]
        warmup_percentage: # warm up percentage
        min_lr: 0.0 # minimum learning rate
        reset_state: False # reset the state of the optimizer
        exponential_config:
            gamma: 0.9
        plateau_config:
            metric: model/train/all/loss
            mode: min
            factor: 0.1
            patience: 10
            threshold: 0.0001
            threshold_mode: rel
            cooldown: 0
            eps: 0.00000001
```

(continues on next page)

(continued from previous page)

```
        step_config:
            step_size: 1
            gamma: 0.1
            last_epoch: -1
        multi_step_config:
            milestones:
                - 1000
            gamma: 0.1
            last_epoch: -1
        cyclic_config:
            base_lr: 0.001
            max_lr: 0.1
            step_size_up: 2000
            step_size_down:
            mode: triangular
            gamma: 1.0
            scale_fn:
            scale_mode: cycle
            cycle_momentum: True
            base_momentum: 0.8
            max_momentum: 0.9
            last_epoch: -1
        one_cycle_config:
            max_lr: 0.1
            pct_start: 0.3
            anneal_strategy: cos
            cycle_momentum: True
            base_momentum: 0.85
            max_momentum: 0.95
            div_factor: 25.0
            final_div_factor: 10000.0
            last_epoch: -1
        cosine_annealing_config:
            last_epoch: -1
    task_scheduler_config:
        task_scheduler: round_robin # [sequential, round_robin, mixed]
        sequential_scheduler_config:
            fillup: False
        round_robin_scheduler_config:
            fillup: False
        mixed_scheduler_config:
            fillup: False
    global_evaluation_metric_dict: # global evaluation metric dict
```

CHAPTER

SIX

LOGGING

This page shows descriptions of the logging functions included with Emmental which logs the learning information and checkpoints.

6.1 Logging Classes

The following docs describe elements of Emmental's logging utilites.

Emmental logging module.

class emmental.logging.**Checkpointer**

Bases: object

Checkpointing class to log train information.

checkpoint(*iteration*, *model*, *optimizer*, *lr_scheduler*, *metric_dict*)

Checkpointing the checkpoint.

Parameters

- **iteration** (Union[float, int]) – The current iteration.
- **model** (*EmmentalModel*) – The model to checkpoint.
- **optimizer** (Optimizer) – The optimizer used during training process.
- **lr_scheduler** (_LRScheduler) – Learning rate scheduler.
- **metric_dict** (Dict[str, float]) – The metric dict.

Return type None

clear()

Clear checkpoints.

Return type None

is_new_best(*metric_dict*)

Update the best score.

Parameters **metric_dict** (Dict[str, float]) – The current metric dict.

Return type Set[str]

Returns The updated best metric set.

load_best_model(*model*)

Load the best model from the checkpoint.

Parameters **model** (*EmmentalModel*) – The current model.

Return type *EmmentalModel*

Returns The best model load from the checkpoint.

class emmental.logging.**JsonWriter**

Bases: *emmental.logging.log_writer.LogWriter*

A class for logging during training process.

add_scalar(*name*, *value*, *step*)

Log a scalar variable.

Parameters

- **name** (str) – The name of the scalar.
- **value** (Union[float, int]) – The value of the scalar.
- **step** (Union[float, int]) – The current step.

Return type None

add_scalar_dict(*metric_dict*, *step*)

Log a scalar variable.

Parameters

- **metric_dict** (Dict[str, Union[float, int]]) – The metric dict.
- **step** (Union[float, int]) – The current step.

Return type None

close()

Close the log writer.

Return type None

write_config(*config_filename='config.yaml'*)

Dump the config to file.

Parameters **config_filename** (str) – The config filename, defaults to "config.yaml".

Return type None

write_log(*log_filename='log.json'*)

Dump the log to file.

Parameters **log_filename** (str) – The log filename, defaults to "log.json".

Return type None

class emmental.logging.**LogWriter**

Bases: object

A class for logging during training process.

add_scalar(*name*, *value*, *step*)

Log a scalar variable.

Parameters

- **name** (str) – The name of the scalar.
- **value** (Union[float, int]) – The value of the scalar.
- **step** (Union[float, int]) – The current step.

Return type None

add_scalar_dict(*metric_dict*, *step*)
Log a scalar variable.

Parameters

- **metric_dict** (Dict[str, Union[float, int]]) – The metric dict.
- **step** (Union[float, int]) – The current step.

Return type None

close()
Close the log writer.

Return type None

write_config(*config_filename='config.yaml'*)
Dump the config to file.

Parameters **config_filename** (str) – The config filename, defaults to "config.yaml".

Return type None

write_log(*log_filename='log.json'*)
Dump the log to file.

Parameters **log_filename** (str) – The log filename, defaults to "log.json".

Return type None

class emmental.logging.**LoggingManager**(*n_batches_per_epoch*, *epoch_count=0*, *batch_count=0*)
Bases: object

A class to manage logging during training progress.

Parameters **n_batches_per_epoch** (int) – Total number batches per epoch.

checkpoint_model(*model*, *optimizer*, *lr_scheduler*, *metric_dict*)
Checkpoint the model.

Parameters

- **model** (*EmmentalModel*) – The model to checkpoint.
- **optimizer** (Optimizer) – The optimizer used during training process.
- **lr_scheduler** (_LRScheduler) – Learning rate scheduler.
- **metric_dict** (Dict[str, float]) – the metric dict.

Return type None

close(*model*)
Close the checkpointer and reload the model if necessary.

Parameters **model** (*EmmentalModel*) – The trained model.

Return type *EmmentalModel*

Returns The reloaded model if necessary

reset()
Reset the counter.

Return type None

trigger_checkpointing()
Check if triggers the checkpointing.

Return type bool

trigger_evaluation()
Check if triggers the evaluation.

Return type bool

update(*batch_size*)
Update the counter.

Parameters **batch_size** (int) – The number of the samples in the batch.

Return type None

write_log(*metric_dict*)
Write the metrics to the log.

Parameters **metric_dict** (Dict[str, float]) – The metric dict.

Return type None

class emmental.logging.**TensorBoardWriter**
Bases: *emmental.logging.log_writer.LogWriter*

A class for logging to Tensorboard during training process.

add_scalar(*name*, *value*, *step*)
Log a scalar variable.

Parameters

- **name** (str) – The name of the scalar.
- **value** (Union[float, int]) – The value of the scalar.
- **step** (Union[float, int]) – The current step.

Return type None

add_scalar_dict(*metric_dict*, *step*)
Log a scalar variable.

Parameters

- **metric_dict** (Dict[str, Union[float, int]]) – The metric dict.
- **step** (Union[float, int]) – The current step.

Return type None

close()
Close the tensorboard writer.

Return type None

write_config(*config_filename='config.yaml'*)
Write the config to tensorboard and dump it to file.

Parameters **config_filename** (str) – The config filename, defaults to "config.yaml".

Return type None

write_log(*log_filename='log.json'*)
Dump the log to file.

Parameters **log_filename** (str) – The log filename, defaults to "log.json".

Return type None

class emmental.logging.**WandbWriter**
Bases: *emmental.logging.log_writer.LogWriter*

A class for logging to wandb during training process.

add_scalar(*name*, *value*, *step*)
Log a scalar variable.

Parameters

- **name** (str) – The name of the scalar.
- **value** (Union[float, int]) – The value of the scalar.
- **step** (Union[float, int]) – The current step.

Return type None

add_scalar_dict(*metric_dict*, *step*)
Log a scalar variable.

Parameters

- **metric_dict** (Dict[str, Union[float, int]]) – The metric dict.
- **step** (Union[float, int]) – The current step.

Return type None

close()
Close the log writer.

Return type None

write_config(*config_filename='config.yaml'*)
Dump the config to file.

Parameters **config_filename** (str) – The config filename, defaults to "config.yaml".

Return type None

write_log(*log_filename='log.json'*)
Dump the log to file.

Parameters **log_filename** (str) – The log filename, defaults to "log.json".

Return type None

6.2 Configuration Settings

Visit the Configuring Emmental page to see how to provide configuration parameters to Emmental via `.emmental-config.yaml`.

The logging parameters of Emmental are described below:

```
# Logging configuration
logging_config:
    counter_unit: epoch # [epoch, batch]
    evaluation_freq: 1
    writer_config:
        writer: tensorboard # [json, tensorboard, wandb]
        verbose: True
```

(continues on next page)

(continued from previous page)

```
        wandb_project_name:
        wandb_run_name:
        wandb_watch_model: False
        wandb_model_watch_freq:
        write_loss_per_step: False
    checkpointing: False
    checkpointer_config:
        checkpoint_path:
        checkpoint_freq: 1
        checkpoint_metric:
            model/train/all/loss: min # metric_name: mode, where mode in [min, max]
        checkpoint_task_metrics: # task_metric_name: mode
        checkpoint_runway: 0 # checkpointing runway (no checkpointing before k unit)
        checkpoint_all: False # checkpointing all checkpoints
        clear_intermediate_checkpoints: True # whether to clear intermediate checkpoints
        clear_all_checkpoints: False # whether to clear all checkpoints
```

CHAPTER

SEVEN

CONFIGURING EMMENTAL

By default, Emmental loads the default config `.emmental-default-config.yaml` from the Emmental directory, and loads the user defined config `emmental-config.yaml` starting from the current working directory, allowing you to have multiple configuration files for different directories or projects. If it's not there, it looks in parent directories. If no file is found, a default configuration will be used.

Emmental will only ever use one `.emmental-config.yaml` file. It does not look for multiple files and will not compose configuration settings from different files.

The default `.emmental-config.yaml` configuration file is shown below:

```
# Meta configuration
meta_config:
    seed: # random seed for all numpy/torch/cuda operations in model and learning
    verbose: True # whether to print the log information
    log_path: logs # log directory
    use_exact_log_path: False # whether to use the exact log directory

# Data configuration
data_config:
    min_data_len: 0 # min data length
    max_data_len: 0 # max data length (e.g., 0 for no max_len)

# Model configuration
model_config:
    model_path: # path to pretrained model
    device: 0 # -1 for cpu or gpu id (e.g., 0 for cuda:0)
    dataparallel: True # whether to use dataparallel or not
    distributed_backend: nccl # what distributed backend to use for DDP [nccl, gloo]

# Learning configuration
learner_config:
    optimizer_path: # path to optimizer state
    scheduler_path: # path to lr scheduler state
    fp16: False # whether to use 16-bit precision
    fp16_opt_level: O1 # Apex AMP optimization level (e.g., ['O0', 'O1', 'O2', 'O3'])
    local_rank: -1 # local_rank for distributed training on gpus
    epochs_learned: 0 # learning epochs learned
    n_epochs: 1 # total number of learning epochs
    steps_learned: 0 # learning steps learned
    n_steps: # total number of learning steps
    skip_learned_data: False # skip learned batches if steps_learned or epochs_learned␣
↪nonzero
```

(continues on next page)

(continued from previous page)

```
    train_split: # the split for training, accepts str or list of strs
        - train
    valid_split: # the split for validation, accepts str or list of strs
        - valid
    test_split: # the split for testing, accepts str or list of strs
        - test
    online_eval: 0 # whether to perform online evaluation
    optimizer_config:
        optimizer: adam # [sgd, adam, adamax, bert_adam]
        parameters: # parameters to optimize
        lr: 0.001 # Learing rate
        l2: 0.0 # l2 regularization
        grad_clip: # gradient clipping
        gradient_accumulation_steps: 1 # gradient accumulation steps
        asgd_config:
            lambd: 0.0001
            alpha: 0.75
            t0: 1000000.0
        adadelta_config:
            rho: 0.9
            eps: 0.000001
        adagrad_config:
            lr_decay: 0
            initial_accumulator_value: 0
            eps: 0.0000000001
        adam_config:
            betas: !!python/tuple [0.9, 0.999]
            eps: 0.00000001
            amsgrad: False
        adamw_config:
            betas: !!python/tuple [0.9, 0.999]
            eps: 0.00000001
            amsgrad: False
        adamax_config:
            betas: !!python/tuple [0.9, 0.999]
            eps: 0.00000001
        lbfgs_config:
            max_iter: 20
            max_eval:
            tolerance_grad: 0.0000001
            tolerance_change: 0.000000001
            history_size: 100
            line_search_fn:
        rms_prop_config:
            alpha: 0.99
            eps: 0.00000001
            momentum: 0
            centered: False
        r_prop_config:
            etas: !!python/tuple [0.5, 1.2]
            step_sizes: !!python/tuple [0.000001, 50]
        sgd_config:
```

(continues on next page)

(continued from previous page)

```
        momentum: 0
        dampening: 0
        nesterov: False
    sparse_adam_config:
        betas: !!python/tuple [0.9, 0.999]
        eps: 0.00000001
    bert_adam_config:
        betas: !!python/tuple [0.9, 0.999]
        eps: 0.00000001
lr_scheduler_config:
    lr_scheduler: # [linear, exponential, reduce_on_plateau, cosine_annealing]
    lr_scheduler_step_unit: batch # [batch, epoch]
    lr_scheduler_step_freq: 1
    warmup_steps: # warm up steps
    warmup_unit: batch # [epoch, batch]
    warmup_percentage: # warm up percentage
    min_lr: 0.0 # minimum learning rate
    reset_state: False # reset the state of the optimizer
    exponential_config:
        gamma: 0.9
    plateau_config:
        metric: model/train/all/loss
        mode: min
        factor: 0.1
        patience: 10
        threshold: 0.0001
        threshold_mode: rel
        cooldown: 0
        eps: 0.00000001
    step_config:
        step_size: 1
        gamma: 0.1
        last_epoch: -1
    multi_step_config:
        milestones:
            - 1000
        gamma: 0.1
        last_epoch: -1
    cyclic_config:
        base_lr: 0.001
        max_lr: 0.1
        step_size_up: 2000
        step_size_down:
        mode: triangular
        gamma: 1.0
        scale_fn:
        scale_mode: cycle
        cycle_momentum: True
        base_momentum: 0.8
        max_momentum: 0.9
        last_epoch: -1
    one_cycle_config:
```

(continues on next page)

(continued from previous page)

```
            max_lr: 0.1
            pct_start: 0.3
            anneal_strategy: cos
            cycle_momentum: True
            base_momentum: 0.85
            max_momentum: 0.95
            div_factor: 25.0
            final_div_factor: 10000.0
            last_epoch: -1
        cosine_annealing_config:
            last_epoch: -1
    task_scheduler_config:
        task_scheduler: round_robin # [sequential, round_robin, mixed]
        sequential_scheduler_config:
            fillup: False
        round_robin_scheduler_config:
            fillup: False
        mixed_scheduler_config:
            fillup: False
    global_evaluation_metric_dict: # global evaluation metric dict

# Logging configuration
logging_config:
    counter_unit: epoch # [epoch, batch]
    evaluation_freq: 1
    writer_config:
        writer: tensorboard # [json, tensorboard, wandb]
        verbose: True
        wandb_project_name:
        wandb_run_name:
        wandb_watch_model: False
        wandb_model_watch_freq:
        write_loss_per_step: False
    checkpointing: False
    checkpointer_config:
        checkpoint_path:
        checkpoint_freq: 1
        checkpoint_metric:
            model/train/all/loss: min # metric_name: mode, where mode in [min, max]
        checkpoint_task_metrics: # task_metric_name: mode
        checkpoint_runway: 0 # checkpointing runway (no checkpointing before k unit)
        checkpoint_all: False # checkpointing all checkpoints
        clear_intermediate_checkpoints: True # whether to clear intermediate checkpoints
        clear_all_checkpoints: False # whether to clear all checkpoints
```

User can also use the Emmental utility function `parse_arg` and `parse_arg_to_config` from `emmental.utils` to generate the config object.

CHAPTER

EIGHT

FREQUENTLY ASKED QUESTIONS (FAQS)

Here are a collection of troubleshooting questions we've seen asked. If you run into anything not covered in this section, feel free to open an Issue.

TBD

CHAPTER

NINE

CHANGELOG

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog and this project adheres to Semantic Versioning 2.0.0 conventions. The maintainers will create a git tag for each release and increment the version number found in emmental/_version.py accordingly. We release tagged versions to PyPI automatically using GitHub Actions.

Note: Emmental is still under active development and APIs may still change rapidly. Until we release v1.0.0, changes in MINOR version indicate backward incompatible changes.

9.1 0.1.1 - 2022-01-11

9.1.1 Fixed

- @lorr1: Fix multiple wand issues. (#118, #119)
- @senwu: Fix scikit-learn version. (#120)

9.2 0.1.0 - 2021-11-24

9.2.1 Deprecated

- @senwu: Deprecated argument *active* in learner and loss function api, and deprecated *ignore_index* argument in configuration. (#107)

9.2.2 Fixed

- @senwu: Fix the metric cannot calculate issue when scorer is none. (#112)
- @senwu: Fix Meta.config is None issue in collate_fn with num_workers > 1 when using python 3.8+ on mac. (#117)

9.2.3 Added

- @senwu: Introduce two new classes: *Action* and *Batch* to make the APIs more modularized and make Emmental more extendable and easy to use for downstream tasks. (#116)

Note: 1. We introduce two new classes: *Action* and *Batch* to make the APIs more modularized.

- *Action* are objects that populate the *task_flow* sequence. It has three attributes: name, module and inputs where name is the name of the action, module is the module name of the action and inputs is the inputs to the action. By introducing a class for specifying actions in the *task_flow*, we standardize its definition. Moreover, *Action* enables more user flexibility in specifying a task flow as we can now support a wider-range of formats for the input attribute of a *task_flow* as discussed in (2).
- *Batch* is the object that is returned from the Emmental *Scheduler*. Each *Batch* object has 6 attributes: uids (uids of the samples), X_dict (input features of the samples), Y_dict (output of the samples), task_to_label_dict (the task to label mapping), data_name (name of the dataset that samples come from), and split (the split information). By defining the *Batch* class, we unify and standardize the training scheduler interface by ensuring a consistent output format for all schedulers.

2. We make the *task_flow* more flexible by supporting more formats for specifying inputs to each module.

- It now supports str as inputs (e.g., inputs="input1") which means take the *input1*'s output as input for current action.
- It also supports a list as inputs which can be constructed by three different formats:
  - x (x is str) where takes whole output of x's output as input: this enables users to pass all outputs from one module to another without having to manually specify every input to the module.
  - (x, y) (y is int) where takes x's y-th output as input.
  - (x, y) (y is str) where takes x's output str as input.

Few emmental.Action examples:

```
from emmental.Action as Act
Act(name="input", module="input_module0", inputs=[("_input_", "data")])
Act(name="input", module="input_module0", inputs=[("_input_", 0)])
Act(name="input", module="input_module0", inputs=["_input_"])
Act(name="input", module="input_module0", inputs="_input_")
Act(name="input", module="input_module0", inputs=[("_input_", "data"), ("_input_", 1), "_
↪input_"])
Act(name="input", module="input_module0", inputs=None)
```

This design also can be applied to action_outputs, here are few example:

```
action_outputs=[(f"{task_name}_pred_head", 0), ("_input_", "data"), f"{task_name}_pred_
↪head"]
action_outputs="_input_"
```

9.3 0.0.9 - 2021-10-05

9.3.1 Added

- @senwu: Support wandb logging. (#99)
- @senwu: Fix log writer cannot dump functions in Meta.config issue. (#103)
- @senwu: Add *return_loss* argument model predict and forward to support the case when no loss calculation can be done or needed. (#105)
- @lorr1 and @senwu: Add *skip_learned_data* to support skip trained data in learning. (#101, #108)

9.3.2 Fixed

- @senwu: Fix model learning that cannot handle task doesn't have Y_dict from dataloasder such as contrastive learning. (#105)

9.4 0.0.8 - 2021-02-14

9.4.1 Added

- @senwu: Support fp16 optimization. (#77)
- @senwu: Support distributed learning. (#78)
- @senwu: Support no label dataset. (#79)
- @senwu: Support output model immediate_ouput. (#80)

Note: To output model immediate_ouput, the user needs to specify which module output he/she wants to output in *EmmentalTask*'s *action_outputs*. It should be a pair of task_flow name and index or list of that pair. During the prediction phrase, the user needs to set *return_action_outputs=True* to get the outputs where the key is *{task_flow name}_{index}*.

```
task_name = "Task1"
EmmentalTask(
    name=task_name,
    module_pool=nn.ModuleDict(
        {
            "input_module": nn.Linear(2, 8),
            f"{task_name}_pred_head": nn.Linear(8, 2),
        }
    ),
    task_flow=[
        {
            "name": "input",
            "module": "input_module",
            "inputs": [("_input_", "data")],
        },
        {
            "name": f"{task_name}_pred_head",
```

(continues on next page)

(continued from previous page)

```
            "module": f"{task_name}_pred_head",
            "inputs": [("input", 0)],
        },
    ],
    loss_func=partial(ce_loss, task_name),
    output_func=partial(output, task_name),
    action_outputs=[
        (f"{task_name}_pred_head", 0),
        ("_input_", "data"),
        (f"{task_name}_pred_head", 0),
    ],
    scorer=Scorer(metrics=task_metrics[task_name]),
)
```

- @senwu: Support action output dict. (#82)
- @senwu: Add a new argument *online_eval*. If *online_eval* is off, then model won't return *probs*. (#89)
- @senwu: Support multiple device training and inference. (#91)

Note: To train model on multiple devices such as CPU and GPU, the user needs to specify which module is on which device in *EmmentalTask*'s *module_device*. It's a ditctionary with key as the module_name and value as device number. During the training and inference phrase, the *Emmental* will automatically perform forward pass based on module device information.

```
task_name = "Task1"
EmmentalTask(
    name=task_name,
    module_pool=nn.ModuleDict(
        {
            "input_module": nn.Linear(2, 8),
            f"{task_name}_pred_head": nn.Linear(8, 2),
        }
    ),
    task_flow=[
        {
            "name": "input",
            "module": "input_module",
            "inputs": [("_input_", "data")],
        },
        {
            "name": f"{task_name}_pred_head",
            "module": f"{task_name}_pred_head",
            "inputs": [("input", 0)],
        },
    ],
    loss_func=partial(ce_loss, task_name),
    output_func=partial(output, task_name),
    action_outputs=[
        (f"{task_name}_pred_head", 0),
        ("_input_", "data"),
        (f"{task_name}_pred_head", 0),
```

(continues on next page)

(continued from previous page)

```
    ],
    module_device={"input_module": -1, f"{task_name}_pred_head": 0},
    scorer=Scorer(metrics=task_metrics[task_name]),
)
```

- @senwu: Add require_prob_for_eval and require_pred_for_eval to optimize score function performance. (#92)

Note: The current approach during score the model will store probs and preds which might require a lot of memory resources especially for large datasets. The score function is also used in training. To optimize the score function performance, this PR introduces two new arguments in *EmmentalTask*: *require_prob_for_eval* and *require_pred_for_eval* which automatically selects whether *return_probs* or *return_preds*.

```
task_name = "Task1"
EmmentalTask(
    name=task_name,
    module_pool=nn.ModuleDict(
        {
            "input_module": nn.Linear(2, 8),
            f"{task_name}_pred_head": nn.Linear(8, 2),
        }
    ),
    task_flow=[
        {
            "name": "input",
            "module": "input_module",
            "inputs": [("_input_", "data")],
        },
        {
            "name": f"{task_name}_pred_head",
            "module": f"{task_name}_pred_head",
            "inputs": [("input", 0)],
        },
    ],
    loss_func=partial(ce_loss, task_name),
    output_func=partial(output, task_name),
    action_outputs=[
        (f"{task_name}_pred_head", 0),
        ("_input_", "data"),
        (f"{task_name}_pred_head", 0),
    ],
    module_device={"input_module": -1, f"{task_name}_pred_head": 0},
    require_prob_for_eval=True,
    require_pred_for_eval=True,
    scorer=Scorer(metrics=task_metrics[task_name]),
)
```

- @senwu: Support save and load optimizer and lr_scheduler checkpoints. (#93)
- @senwu: Support step based learning and add argument *start_step* and *n_steps* to set starting step and total step size. (#93)

9.4.2 Fixed

- @senwu: Fix customized optimizer support issue. (#81)
- @senwu: Fix loss logging didn't count task weight. (#93)

9.5 0.0.7 - 2020-06-03

9.5.1 Added

- @senwu: Support gradient accumulation step when machine cannot run large batch size. (#74)
- @senwu: Support user specified parameter groups in optimizer. (#74)

Note: When building the emmental learner, user can specify parameter groups for optimizer using *emmental.Meta.config["learner_config"]["optimizer_config"]["parameters"]* which is function takes the model as input and outputs a list of parameter groups, otherwise learner will create a parameter group with all parameters in the model. Below is an example of optimizing Adam Bert.

```
def grouped_parameters(model):
    no_decay = ["bias", "LayerNorm.weight"]
    return [
        {
            "params": [
                p
                for n, p in model.named_parameters()
                if not any(nd in n for nd in no_decay)
            ],
            "weight_decay": emmental.Meta.config["learner_config"][
                "optimizer_config"
            ]["l2"],
        },
        {
            "params": [
                p
                for n, p in model.named_parameters()
                if any(nd in n for nd in no_decay)
            ],
            "weight_decay": 0.0,
        },
    ]

emmental.Meta.config["learner_config"]["optimizer_config"][
    "parameters"
] = grouped_parameters
```

9.5.2 Changed

- @senwu: Enabled "Type hints (PEP 484) support for the Sphinx autodoc extension." (#69)
- @senwu: Refactor docstrings and enforce using flake8-docstrings. (#69)

9.6 0.0.6 - 2020-04-07

9.6.1 Added

- @senwu: Support probabilistic gold label in scorer.
- @senwu: Add *add_tasks* to support adding one task or mulitple tasks into model.
- @senwu: Add *use_exact_log_path* to support using exact log path.

Note: When init the emmental there is one extra argument *use_exact_log_path* to use exact log path.

```
emmental.init(dirpath, use_exact_log_path=True)
```

9.6.2 Changed

- @senwu: Change running evaluation only when evaluation is triggered.

9.7 0.0.5 - 2020-03-01

9.7.1 Added

- @senwu: Add *checkpoint_all* to controll whether to save all checkpoints.
- @senwu: Support *CosineAnnealingLR*, *CyclicLR*, *OneCycleLR*, *ReduceLROnPlateau* lr scheduler.
- @senwu: Support more unit tests.
- @senwu: Support all pytorch optimizers.
- @senwu: Support accuracy@k metric.
- @senwu: Support cosine annealing lr scheduler.

9.7.2 Fixed

- @senwu: Fix multiple checkpoint_metric issue.

9.8 0.0.4 - 2019-11-11

9.8.1 Added

- @senwu: Log metric dict into log file every trigger evaluation time or full epoch.
- @senwu: Add *get_num_batches* to calculate the total number batches from all dataloaders.
- @senwu: Add *n_batches* in *EmmentalDataLoader* and *fillup* in *Scheduler* to support customize dataloader.
- @senwu: Add overall and task specific loss during evaluating as default. to support user needs for clear checkpoins.
- @senwu: Add *min_len* and *max_len* in *Meta.config* to support setting sequence length.
- @senwu: Add overall and task specific loss during evaluating as default.
- @senwu: Calculate overall and task specific metrics and loss during training.
- @senwu: Add more util functions, e.g., array_to_numpy, construct_identifier, and random_string.
- @senwu: Enforce dataset has uids attribute.
- @senwu: Add micro/macro metric options which have split-wise micro/macro average and global-wise micro/macro average. The name for the metrics are:

```
split-wise micro average: `model/all/{split}/micro_average`
split-wise macro average: `model/all/{split}/macro_average`
global-wise micro average: `model/all/all/micro_average`
global-wise macro average: `model/all/all/macro_average`
```

***Note*: *micro* means average all metrics from all tasks. *macro* means average all** average metric from all tasks.

- @senwu: Add contrib folder to support unofficial usages.

9.8.2 Fixed

- @senwu: Correct lr update for epoch-wised scheduler.
- @senwu: Add type for class.
- @senwu: Add warning for one class in ROC AUC metric.
- @senwu: Fix missing support for StepLR and MultiStepLR lr scheduler.
- @senwu: Fix missing pytest.ini and fix test cannot remove temp dir issue.
- @senwu: Fix default train loss metric from *model/train/all/loss* to *model/all/train/loss* to follow the format *TASK_NAME/DATA_NAME/SPLIT/METRIC* pattern.

9.8.3 Changed

- @senwu: Change default grad clip to None.
- @senwu: Update seed and grad_clip to nullable.
- @senwu: Change default class index to 0-index.
- @senwu: Change default ignore_index to None.
- @senwu: Change the default counter unit to epoch.
- @senwu: Update the metric to return one metric value by default.

9.8.4 Removed

- @senwu: Remove *checkpoint_clear* argument.

CHAPTER

TEN

INSTALLATION

To test changes in the package, you install it in editable mode locally in your virtualenv by running:

```
$ make dev
```

This will also install our pre-commit hooks and local packages needed for style checks.

Tip: If you need to install a locally edited version of emmental in a separate location, such as an application, you can directly install your locally modified version:

```
$ pip install -e path/to/emmental/
```

in the virtualenv of your application.

CHAPTER

ELEVEN

TESTING

We use pytest to run our tests. Our tests are all located in the `tests` directory in the repo, and are meant to be run *after* installing Emmental locally.

To run our tests, just run:

```
$ make test
```

CHAPTER

TWELVE

CODE STYLE

For code consistency, we have a pre-commit configuration file so that you can easily install pre-commit hooks to run style checks before you commit your files. You can setup our pre-commit hooks by running:

```
$ pip install -r requirements-dev.txt
$ pre-commit install
```

Or, just run:

```
$ make dev
```

Now, each time you commit, checks will be run using the packages explained below.

We use black as our Python code formatter with its default settings. Black helps minimize the line diffs and allows you to not worry about formatting during your own development. Just run black on each of your files before committing them.

Tip: Whatever editor you use, we recommend checking out black editor integrations to help make the code formatting process just a few keystrokes.

For sorting imports, we reply on isort. Our repository already includes a *.isort.cfg* that is compatible with black. You can run a code style check on your local machine by running our checks:

```
$ make check
```

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX

I

J

L

M

P

R

S

T

U

W