panna Documentation

Release prerelease

pannadevs

Nov 15, 2018

CONTENTS:

Automatically created by Sphinx using the documentation in the source code

CHAPTER

ONE

SIMULATION - THE DATA

class simulation.**Example**(*g_vectors*, *species_vector*, *true_energy*, *zeros=None*, *atomic_species=None*, *name=None*)

Example class

simulation.**iterator_over_tfdata**(*g_size*, **args*, ***kwargs*)

TFdata unpacker

Parameters

- **g_size** – size of the G's
- **args** – all the tfdata files that one wants to parse

kwargs: zeros: list of zeros, one per species

Retrun: iterator over the record in the files

CHAPTER

TWO

INPUT OF THE NETWORK

Utilities to handling the input system

`inputs.`**`parse_fn_v1`**`(`*example*, *g_size*, *zeros*, *n_species*`)`

Parse TFExample records and perform simple data augmentation.

Parameters

- **`example`** – a batch of example obj
- **`g_size`** – size of the g_vector
- **`zeros`** – array of zero's one value per specie.
- **`n_species`** – number of species

Returns Sparse Tensor, (n_atoms) value in range(n_species) g_vectors_tensor: Sparse Tensor, (n_atoms, g_size) energy: true energy value corrected with the zeros

Return type species_tensor

`inputs.`**`input_iterator`**`(`*data_dir*, *batch_size*, *parse_fn*, *name*, *shuffle_buffer_size_multiplier=10*, *prefetch_buffer_size_multiplier=20*, *num_parallel_readers=8*, *num_parallel_calls=8*, **args*, *oneshot=None*`)`

Construct input iterator.

Parameters

- **`data_dir`** – directory for data, must contain a "train_tf subfolder"
- **`batch_size`** – batch size
- **`parse_fn`** – function to parse the data from tfrecord file
- **`name`** – name scope
- **`*_buffer_size_multiplier`** – batchsize times this number
- **`num_parallel_readers`** – process that are doing Input form drive
- **`num_parallel_calls`** – call of the parse function
- **`oneshot`** – experimental, do not set
- **`TODO`** – construct a double system to handle in_place evaluation of accuracy

Returns initializable_iterator, recover input data to feed the model

Note:

- shuffling batch and buffer size multiplier default are randomly chosen by me

- initializable iterator can be changed to one shot iterator in future version to better comply with documentation
- a maximum number of epoch should also be added to this routine.

CHAPTER

THREE

CHECKPOINT: SAVING AND RESTART ROUTINE

class checkpoint.**Checkpoint**(*filename*, *atoms_list=None*)

class to handle a Checkpoint

class checkpoint.**Parameters**(*file_name*, *atoms_list=None*)

A class that load parameters form files generated by and is compatible with the Network object

CHAPTER

FOUR

INPUT PARSER

parser_callable.**get_network_architecture**(*value*)

parse the architecture format

Parameters

- **value** – string like
- **layer_size** – layer2_size…

Returns list of size per layer

parser_callable.**get_network_trainable**(*value*)

parse trainable list

Parameters

- **value** – string like
- **1** – 0:1

Returns list of trainable flag per layer

CHAPTER FIVE

NETWORK ARCHITECTURE

networks.**network_A2A**(*batch_of_species*, *batch_of_gvects*, *layer_size*, *trainability*, *gvect_size*, *batch_size*, *Nspecies*, *atomic_label*, *import_layer=None*, *reuse=None*)

New network with variable architecture annd species resolved weights.

Parameters

- **batch_of_species** – [batch size x max number of atoms per molecule] species vector for each element of the batch
- **batch_of_gvects** – batch of gvectors
- **layer_size** – a list of lists with the size of each hidden layer, for each species
- **trainability** – a list of lists with the boolean flag of the trainable status of each hidden layer, for each species
- **gvect_size** – gvector size
- **batch_size** – number of calculations in a batch
- **Nspecies** – number of species
- **import_layer** – a list of lists of tuple, each tuple has 2 elements, weights and biases that can be either a tensor with correct shape or None
- **reuse** – whether to reuse variables with the same name

Returns

tf.tensor of energies natoms_batch: tf.tensor of number of atoms for each element of the

batch

Return type Energy

networks.**loss_NN**(*batch_energies*, *batch_energies_dft*, *batch_natoms*)

this is simply our cost function

Parameters

- **= prediction of the network** (*batch_energies*) –
- **= energies labels** (*batch_energies_dft*) –
- **= number of atoms** (*batch_natoms*) –

Returns the loss value tensor with delta_e for each element of the batch

class networks.**eval_network_A2A**(*checkpoint*)

A2A network implementation

CHAPTER

SIX

REGULARIZATIONS

`regularizations.`**`l1l2_regularizations`**`(`*wscale_l1*, *wscale_l2*, *bscale_l1*, *bscale_l2*`)`

Apply required regularizator.

Parameters

- **`w`** – weights, b : biases
- **`l1`** – norm one prefactor if zero nothing gets applyed
- **`l2`** – norm two prefactor if zero nothing gets applyed

Returns A scalar representing the overall regularization penalty for W A scalar representing the overall regularization penalty for B

CHAPTER

SEVEN

LAYERS OF THE NETWORK

train_ops.**train_NN**(*loss*, *global_step*, *lr*, *atomic_sequence*)

Train NN model, optimization step.

Create an optimizer and apply to all trainable variables. Add moving average for all trainable variables.

Parameters

- **loss** – quantity to minimize
- **global_step** – Integer Variable counting the number of training steps processed.
- **lr** – learning rate
- **atomic_sequence** – just for now here to simplify creation of histogram…

Returns op for training.

Return type train_op

TODO: refactor this routine…. it is too big and does too many stuff

CHAPTER

EIGHT

REAL LAYERS OF THE NETWORK

`layers.`**`hidden_layer_gauss`**(*in_tensor*, *in_size*, *out_size*, *trainable*, *init_values=(None*, *None)*)

Define an all to all connected layer with species division and Gaussian activation function.

Parameters

- **`in_tensor`** – input to be computed,
- **`in_size`** – last dimension of the input,
- **`out_size`** – last dimension of the output,
- **`trainable`** – whether we should train these weights
- **`init_values`** – numpy arrays to initialize the tensors, weights and biases None = default initialization

Returns Output of the layer

weights variable will be named “weights” bias variable will be named “bias”

`layers.`**`hidden_layer_linear`**(*in_tensor*, *in_size*, *out_size*, *trainable*, *init_values=(None*, *None)*)

Define an all to all connected layer with species division and linear activation function. TODO: Make a single layer that accepts activation function

Parameters

- **`in_tensor`** – input to be computed,
- **`in_size`** – last dimension of the input,
- **`out_size`** – last dimension of the output,
- **`trainable`** – whether we should train these weights
- **`init_values`** – numpy arrays to initialize the tensors, weights and biases None = default initialization

Returns Output of the layer

weights variable will be named “weights” bias variable will be named “bias”

CHAPTER

NINE

TF VARIABLE HELPERS

These are specifically for TF variables .. automodule:: variable_helpers .. autofunction:: _variable_on_cpu .. autofunction:: _variable_random_uniform

CHAPTER

TEN

RELEASE NOTES & TO-DO

Here can be release notes change log and to do

Todo: Update the below link when we add new guides on these.

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX