Grapa Documentation

Release 0.1

Simon Klüttermann

Oct 25, 2020

Contents

A simple python3 library using keras and tensorflow to do Deep Learning on Graphs, with a strong emphasis on Codes that chance the Shape of the Graph in a way that can be used for Autoencoder

We split our Code into two different Apis: a low level Layer Api, and a high Level Functional Api. We also provide 5 different example Applications to get you started using grapa as fast as possible.

Contents:

CHAPTER 1

getting started

We show you 5 different Applications, that should make it easy to work with grapa for everything you migth want to use it for

Contents:

1.1 Top Tagging

This module is based on a master thesis about top tagging. So our first application is simply this.

1.2 Nets

QQQ

1.3 mol

QQQ

1.4 feyn

QQQ

1.5 tipsy

QQQ

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

Layer Api

The Layer Api contains many low level layers. This means that you can do much more than using the Functional Api, but whatever you do, migth be easier achieved by the Functional Api.

Contents:

3.1 gadd1

Takes a Adjacency Matrix and adds an Identity to it

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)

3.2 gaddbias

Takes a Feature Vector and adds a bias to it

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node
- **metrik_init="glorot_uniform"**: initializer of the bias
- **learnabel=True**: Shall the Bias be learnable

3.3 gaddzeros

Takes a Feature Vector and creates some empty (zero) nodes

Arguments

- **inn**: The initial Number of Nodes of the Graph (Graph Size)
- **out**: The output Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node

3.4 gbrokengrowth

Easy way to increase the number of nodes in a Feature Vector. Uses a single Dense Layer to transform every Feature in those that are new. Breaks the Graph Permutation Symmetry.

Arguments

- **inn**: The initial Number of Nodes of the Graph (Graph Size)
- **out**: The final Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node

3.5 glbuilder

Similar to gbuilder and gkeepbuilder, this builds a Graph from a Feature Vector and concattes them

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the first input Vector
- **free**: Additional Features that are createt by this Layer
- **metrik_initializer=metrik_init**: Initializer of the Metrik of this Layer. Defaults to metrik_init which returns a 0 Matrix, with an 1 in the 4,5 (counting from 0) diagonal Entry.

3.6 gchooseparam

Transforms a Feature Vector into one with just q elements

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node
- **q=[0,3]**: which features to keep

3.7 gcomdensediverge

Inputs a 3d Vector of dimensions (-1,gs,param) and uses a simple dense Layer to transform it into (-1,gs,c,paramo) without breaking Graph Permutation Symmetry. So transforms each Feature Vector into a set of c Feature Vectors.

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node

- **paramo**: The Number of Features each Outputvector should have
- **c=2**: The Number of Outputvectors for each Inputfeaturevector
- **initializer=glorot_uniform**: THe Transformation initializer
- **learnabel=True**: Shall the Transformation be learnable

3.8 gcomdensemerge

Takes a 4d Tensor (-1,gs,ags,param) and transforms it using a simple Dense Layer into a 3d one (-1,gs,paramo). Respects Graph permutation Symmetry. So Basically Transforms each set of vectors into one vector of different size

Arguments

- **gs**: The Number of Lists of Feature Vectors and the Number of Nodes in the Output
- **ags**: The Number of Vectors in each List of Input vector, could be understood as the opposite of c
- **param**: The Number of Features for each Input Node
- **paramo**: The Number of Features each Outputvector should have
- **initializer=glorot_uniform**: The Transformation initializer
- **learnabel=True**: Shall the Transformation be learnable

3.9 gcomdepoollg

Extension of gcomdepool by also learning a Graph. Breaks Graph Permutation Symmetry, by using a Dense layer to create a c*c Adjacency Matrix from all features. Returns the Feature Vectors and the Adjacency Matrices

Arguments

- **gs**: The initial Number of Nodes of the Graph (Graph Size)
- **param**: The initial Number of Features for each node
- **paramo**: The Number of Features each Outputvector should have
- **c=2**: The Factor by which to increase gs
- **metrik_init=glorot_uniform**: The Transformation initializer to go from little Feature Vector to output Vectors
- **graph_init=keras.initializers.Identity()**: The Initializer to create the Graph
- **learnable=True**: Shall the Transformation be learnable

3.10 gcomdepoolplus

Similar to gcomdepoollg, but the Graph is not a function of the Features, but a constant

Arguments

- **gs**: The initial Number of Nodes of the Graph (Graph Size)
- **param**: The initial Number of Features for each node
- **paramo**: The Number of Features each Outputvector should have

- **c=2**: The Factor by which to increase gs
- **metrik_init=glorot_uniform**: The Transformation initializer to go from little Feature Vector to output Vectors
- **graph_init=keras.initializers.Identity**(): The Initializer to create the Graph
- **learnable=True**: Shall the Transformation be learnable

3.11 gcomdepool

Transforms a 3d Vector (-1,gs,param) into a 4d vector (-1,gs,c,paramo) using a Dense Layer. It respects Graph Permutation Symmetry by simply transforming each Feauture Vector into a List of c Feature Vectors

Arguments

- **gs**: The initial Number of Nodes of the Graph (Graph Size)
- **param**: The initial Number of Features for each node
- **paramo**: The Number of Features each Outputvector should have
- **c=2**: The Factor by which to increase gs
- **metrik_init=glorot_uniform**: The Transformation initializer to go from little Feature Vector to output Vectors
- **learnable=True**: Shall the Transformation be learnable

3.12 gcomdex

Takes a Feature Vector and returns the Indices of this Feautre Vector in the node dimension in the Order of the last Feature (desc). Returned Integers are cast to float32 to keep the numeric types consistent.

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node

3.13 gcomdiagraph

Takes a Adjacency Matrix of Size (-1,gs,gs) and cuts out the diagonal Parts of it, here each diagonal entry has size (c,c), so that the result has the shape (-1,gs/c,c,c).

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size). Has to be a multiple of c
- **c**: The Number of Nodes in each subgraph

3.14 gcomextractdiag

Takes a 5 dimensional Tensor of size (-1,gs,gs,c,c) and transforms it into a 4d Tensor of size (-1,gs,c,c) by demanding that the second and third index are the same (A_ijklm into A_ijjlm)

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size).
- **c**: The Number of Nodes in each subgraph

3.15 gcomfullyconnected

Creates a fully connected Graph of size (-1,gs,gs) (so basically just a 1). Uses a Feature vector to find the size of the first dimension

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size).
- **param**: The Number of Features in each Node. Here only used for consistency checks

3.16 gcomgpool

Takes an Adjacency Matrix and a Feature Vector, orders the Feature Vector by its last Feature and the Adjacency Matrix in the same way. Then uses a learnable Transformation to go from each neighbouring c vectors of size param into one vector of size paramo. and uses mode to go from a big adjacency Matrix of size gs to one of size gs, after which it roundes the resulting Adjacency Matrix to either 0 or 1 and returns both the new Adjacency Matrix and the new Feature Vector. Basically a simpler (and not used by me) version of the function abstr

Arguments

- **gs**: The initial Number of Nodes of the Graph (Graph Size).
- **param**: The initial Number of Features in each Node.
- **paramo**: The resulting Number of Features in each Node.
- **metrik_init="glorot_uniform"**: The initializer of the Feautre Vector tranformation
- **learnable=True**: weather the Transformation is learnable
- **mode="mean"**: Either "mean", "max" or "min", The Transformation to go combine each subgraph into a single node
- **cut=0.5**: Where to round the graph after the mode transformation, Each Value above cut will be set to 1 and each value below to 0, each Value that is equal to cut will be also set to 1
- **c_const=1000**: Since the Rounding is implementet using relus (relu(1+c_const*(X-cut))-relu(c_const*(X-cut))) , you require a numerical constant to decide how fast the rounding function goes from 0 to 1. This could be set higher, but this could result in diverging gradients.

3.17 gcomgraphand2

Takes Two Adjacency Matrices of size c*gs and combines them in a way defined by mode. Here by setting the graph to size c*gs instead of gs is done just for consitency in a function

Arguments

- **c*gs**: The Number of Nodes of each Graph.
- **mode**: either "and","prod","or" or "sum". Here "and" and "or" are Rounded versions of "prod" and "sum" respectively. The kind of Transformation that combines the Graphs.

- **cut=0.5**: Where to round the graph after the mode transformation, Each Value above cut will be set to 1 and each value below to 0, each Value that is equal to cut will be also set to 1
- **c_const=1000**: Since the Rounding is implementet using relus (relu(1+c_const*(X-cut))-relu(c_const*(X-cut))) , you require a numerical constant to decide how fast the rounding function goes from 0 to 1. This could be set higher, but this could result in diverging gradients.

3.18 gcomgraphand

Similar to gcomgraphand2 (yes the naming migth be a bit confusing). Here takes an Tensor of size (-1,gs,gs,c,c,n) resulting in (-1,gs,gs,c,c). So it runs the same kind of mode Transformation on a graph of n graphs (gs, c) instead of 2 graphs.

Arguments

- **gs**: The Number of Nodes in each global Graph.
- **c=2**: The Number of Nodes in each subgraph.
- **n=2**: The Number of Graphs that are combines using mode
- **mode**: either “and”,”prod”,”or” or “sum”. Here “and” and “or” are Rounded versions of “prod” and “sum” respectively. The kind of Transformation that combines the Graphs.
- **cut=0.5**: Where to round the graph after the mode transformation, Each Value above cut will be set to 1 and each value below to 0, each Value that is equal to cut will be also set to 1
- **c_const=1000**: Since the Rounding is implementet using relus (relu(1+c_const*(X-cut))-relu(c_const*(X-cut))) , you require a numerical constant to decide how fast the rounding function goes from 0 to 1. This could be set higher, but this could result in diverging gradients.

3.19 gcomgraphcombinations

Takes a 4d Layer os size (-1,gs,c,c) and concattes every possible combination in the first nonbatch dimension (-1,gs,gs,c,c,2)

Arguments

- **gs**: The size of the combination dimensions.
- **c**: The size of each subgraph

3.20 gcomgraphcutter

Takes a Adjacency Matrix of size gs, cuts it into c subgraphs, and uses mode Transformation to combine those c subgraphs and rounded the Grahs afterwards.

Arguments

- **gs**: The initial Number of Nodes in the Graph.
- **c=2**: The Number of subgraphs, must gs must divide it.
- **mode**: either “mean”,”min” or “max”. The kind of Transformation that combines the Graphs.
- **cut=0.5**: Where to round the graph after the mode transformation, Each Value above cut will be set to 1 and each value below to 0, each Value that is equal to cut will be also set to 1

- **c_const=1000**: Since the Rounding is implementet using relus (relu(1+c_const*(X-cut))-relu(c_const*(X-cut))) , you require a numerical constant to decide how fast the rounding function goes from 0 to 1. This could be set higher, but this could result in diverging gradients.

3.21 gcomgraphfrom2param

Takes a 5d Tensor of size (-1,gs,gs,param,n) and transforms it into (-1,gs,gs,c,c). So basically creates a Graph of size c for each set of n param vectors, using a simple Dense Layer to do the (param,n) to (c,c) transformation.

Arguments

- **gs**: The Number of Nodes in each global Graph.
- **param**: The Number of Features from which the graphs will be constructed
- **c=2**: The Number of Nodes in each resulting subgraph.
- **n=2**: The Number of Feature Vectors that result in each subgraph
- **initializer="glorot_uniform"**: The initializer of the Transformation
- **trainable=True**: weather the Transformation is trainable

3.22 gcomgraphfromparam

similar to gcomgraphfrom2param but for a 3d Tensor instead of a 5d Tensor ((-1,gs,param) to (-1,gs,c,c))

Arguments

- **gs**: The Number of Nodes in each global Graph.
- **param**: The Number of Features from which the graphs will be constructed
- **c=2**: The Number of Nodes in each resulting subgraph.
- **initializer="glorot_uniform"**: The initializer of the Transformation
- **trainable=True**: weather the Transformation is trainable

3.23 gcomgraphlevel

Converts a 5d Tensor of size (-1,gs,gs,c,c) into (-1,c*gs,c*gs)

Arguments

- **gs**: The Number of Nodes in each global Graph.
- **c=2**: The Number of Nodes in each subgraph.

3.24 gcomgraphlevel

Converts a 3d Tensor (-1,gs,gs) into (-1,c*gs,c*gs) by repetition

Arguments

- **gs**: The Number of Nodes in each Graph.

- **c=2**: The Number of repetition this Layer should do.

3.25 gcomjpool

Takes a Feature Vector of size (-1,gs,param) and some indices representing order indices (-1,gs) and uses these indices to reorders the Feature Vectors and applies a simple Dense Transformation to transform it into (-1,gs/c,paramo)

Arguments

- **gs**: The initial Number of Nodes in each Graph.
- **param**: The initial Number of Features.
- **c=2**: How many Feature Vectors to combine into each Outputvector. gs has to divide this
- **paramo**: The Number of Features in each Outputvector
- **metrik_init="glorot_uniform"**: The initializer of the Transformation
- **trainable=True**: weather the Transformation is trainable

3.26 gcomparamcombinations

Takes a 3d Vector (-1,gs,param) and returns each possible Combination in gs (-1,gs,gs,param,2)

Arguments

- **gs**: The Number of Nodes in each Graph.
- **param**: The Number of Features in each Node.

3.27 gcomparamlevel

Takes a 4 Tensor of size(-1,gs,c,param) and levels it down into (-1,c*gs,param)

Arguments

- **gs**: The Number of Nodes in each global Graph.
- **c**: The Number of Nodes in each subgraph.
- **param**: The Number of Features in each Node.

3.28 gcomparastract

Cuts a Feature Vector of size (-1,gs,param) into (gs/c)*c Feature Vectors (-1,gs/c,c,param)

Arguments

- **gs**: The initial Number of Nodes in each Graph.
- **c**: The Size (Number of Nodes) of the Feature Vectors that will be cut out of the Input.
- **param**: The Number of Features in each Node.

3.29 gcompoolmerge

Takes a 4d vector of size (-1,gs,ags,param) and uses mode to transform it into (-1,gs,param)

Arguments

- **gs**: The Number of Nodes in each Graph.
- **ags**: The Number of Nodes in each subgraph.
- **mode="max"**: The Transformation that will be used, has to be either "max", "min" or "mean"
- **param**: The Number of Features in each Node.

3.30 gcompool

Orders a Feature vector of size (-1,gs,param), orders it by its last Feature and uses a simple Dense Layer to transform each c neighbouring Feature Vectors into one Feature Vector of size paramo (-1,gs/c,paramo)

Arguments

- **gs**: The initial Number of Nodes in each Graph.
- **param**: The initial Number of Features in each Node.
- **paramo**: The Number of Features in each output Node.
- **c=2**: How many Input Feature Vectors are to be transformed into one Output Vector
- **metrik_init="glorot_uniform"**: Initializer of the Transformation
- **learnable=True**: Weather the Transformation is learnable.

3.31 gcomreopool

Takes an Adjacency Matrix and a Feature Vector and orders them by the last Feauture Index

Arguments

- **gs**: The Number of Nodes in each Graph.
- **param**: The Number of Features in each Node.

3.32 gcutparam

Splits a Feature Vector (-1,gs,param1+param2) into two Feature Vectors (-1,gs,param1) and (-1,gs,param2). Opposite of ghealparam.

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param1**: The Number of Features for each node in the first output Vector
- **param2**: The Number of Features for each node in the second output Vector

3.33 gcutter

Cuts the last Nodes from a Featurevector (-1,inn,param) => (-1,out,param)

Arguments

- **inn**: The initial Number of Nodes of the Graph (Graph Size)
- **out**: The output Number of Nodes of the Graph, has to be at most as big as inn
- **param**: The Number of Features in each Feature vector

3.34 gecutter

Like gcutter, but leaves not the first but the last out nodes

Arguments

- **inn**: The initial Number of Nodes of the Graph (Graph Size)
- **out**: The output Number of Nodes of the Graph, has to be at most as big as inn
- **param**: The Number of Features in each Feature vector

3.35 gfeatkeep

If your Adjacency Matrix or (dimension+1) Adjacency Matrices are concattet, this can cut out the Feature Vector. Extension of gfeat

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features in each Feature vector
- **dimension=0**: The Number of Adjacency Matrices concattet together (ignoring the first one) before the features

3.36 gfeat

If your Adjacency Matrix is concattet to the Feature Vector, this can cut out the Feature Vector

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features in each Feature vector
- **dimension=0**: The Number of Adjacency Matrices concattet together (ignoring the first one) before the features

3.37 gfromparam

Transforms a 3 dimensional Featurevector (-1,gs,param) into a 2 dimensional Featurevector (-1,gs*param) that can be used for example with classical Dense Layers. Opposite of ggoparam

Arguments

- **gs=1**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features in each Feature vector

3.38 ggoparam

Transforms a 2 dimensional Featurevector (-1,gs*param) into a 3 dimensional Featurevector (-1,gs,param). Opposite of gfromparam

Arguments

- **gs=1**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features in each Feature vector

3.39 ggraphstract

Combines two Adjacency Matrices of size inn1 and inn2 respectively into one Adjacency Matrix of size inn1*inn2 by calculating the Kronecker Produkt Both Inputs can have a Batch Dimension (-1,inn,inn) or not (inn,inn)

Arguments

- **in1**: The first Adjacency Matrix
- **in2**: The second Adjacency Matrix

3.40 ghealparam

Combines two Feature Vectors (-1,gs,param1) and (-1,gs,param2) into one Feature Vector (-1,gs,param1+param2). Opposite of gcutparam.

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param1**: The Number of Features for each node in the first input Vector
- **param2**: The Number of Features for each node in the second input Vector

3.41 gkeepbuilder

Extension of gbuilder, allowing for multiple Adjacency Matrices (dimension+1) and a learnable metrik.

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the first input Vector
- **free**: Additional Features that are createt by this Layer
- **learnable=True**: Is the metrik learnable?
- **dimension=0**: Number of Additional Adjacency Matrices
- **use0=False**:Allows you to toggle, if your metrik and thus distance should include the first of the variables

3.42 gkeepcutter

Cuts a concattet Graph with (dimension+1) Adjacency Matrices of size inn into a Graph of size out

Arguments

- **inn**: The initial Number of Nodes of the Graph (Graph Size)
- **out**: The resulting Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the first input Vector
- **dimension=0**: Number of Additional Adjacency Matrices

3.43 gkeepmatcut

Cuts out the first Adjacency Matrix out of a Concattet Graph with (dimension+1) Adjacency Matrices

Arguments

- **gs**: The gs Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the first input Vector
- **dimension=0**: Number of Additional Adjacency Matrices

3.44 glacreate

Extension of glcreate to work also on a abstract data.

Arguments

- **gs**: The gs Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the Feature Vector
- **a=2**: Size of the abstraction, think of this as the size of a second batch simension

3.45 glam

Extension of glm to work with abstract data Inputs an Adjacency matrix and a Feature vector, and returns the updated Feature vector

Arguments

- **gs**: The gs Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the Feature Vector
- **a**: Size of the abstraction, think of this as the size of a second batch simension
- **iterations=1**: repeat the Actions of this Layer iterations time
- **alinearity=[-1.0,1.0]**: activation of this Layer, explained better in glm
- **kernel_initializer="glorot_uniform"**: Initializer of this Layer
- **self_initializer=None**: Instead of using kernel_initializer, this can be used to specify an initializer just for the self interaction of this Layer. Has preference over kernel_initializer.

- **neig_initializer=None**: Instead of using kernel_initializer, this can be used to specify an initializer just for the neighbour interaction of this Layer. Has preference over kernel_initializer
- **learnable=True**: weather this Layer has learnable Variables (self and neighbour interaction). Useful for debugging sometimes

3.46 gliam

Extension of glim to work with abstract data. Inverts an equivalent glam. Inputs an Adjacency matrix and a Feature vector, and returns the updated Feature vector

Arguments

- **gs**: The gs Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the Feature Vector
- **a**: Size of the abstraction, think of this as the size of a second batch simension
- **iterations=1**: repeat the Actions of this Layer iterations time
- **alinearity=[-1.0,1.0]**: activation of this Layer, explained better in glm
- **kernel_initializer="glorot_uniform"**: Initializer of this Layer
- **self_initializer=None**: Instead of using kernel_initializer, this can be used to specify an initializer just for the self interaction of this Layer. Has preference over kernel_initializer.
- **neig_initializer=None**: Instead of using kernel_initializer, this can be used to specify an initializer just for the neighbour interaction of this Layer. Has preference over kernel_initializer
- **learnable=True**: weather this Layer has learnable Variables (self and neighbour interaction). Useful for debugging sometimes

3.47 glim

A Layer that inverts a glm Layer with the same Variables. Inputs an Adjacency matrix and a Feature vector, and returns the updated Feature vector

Arguments

- **gs**: The gs Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the Feature Vector
- **a**: Size of the abstraction, think of this as the size of a second batch simension
- **iterations=1**: repeat the Actions of this Layer iterations time
- **alinearity=[-1.0,1.0]**: activation of this Layer, explained better in glm
- **kernel_initializer="glorot_uniform"**: Initializer of this Layer
- **self_initializer=None**: Instead of using kernel_initializer, this can be used to specify an initializer just for the self interaction of this Layer. Has preference over kernel_initializer.
- **neig_initializer=None**: Instead of using kernel_initializer, this can be used to specify an initializer just for the neighbour interaction of this Layer. Has preference over kernel_initializer
- **learnable=True**: weather this Layer has learnable Variables (self and neighbour interaction). Useful for debugging sometimes

3.48 glkeep

Version of glm to handle concatted Graphs. Works with the first of (dimension+1) Adjacency Matrices.

Arguments

- **gs**: The gs Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the Feature Vector
- **a**: Size of the abstraction, think of this as the size of a second batch simension
- **iterations=1**: repeat the Actions of this Layer iterations time
- **alinearity=[-1.0,1.0]**: activation of this Layer, explained better in glm
- **kernel_initializer="glorot_uniform"**: Initializer of this Layer
- **self_initializer=None**: Instead of using kernel_initializer, this can be used to specify an initializer just for the self interaction of this Layer. Has preference over kernel_initializer.
- **neig_initializer=None**: Instead of using kernel_initializer, this can be used to specify an initializer just for the neighbour interaction of this Layer. Has preference over kernel_initializer
- **learnable=True**: weather this Layer has learnable Variables (self and neighbour interaction). Useful for debugging sometimes
- **dimension=0**: Number of additional Adjacency Matrices

3.49 glmlp

Extension of gltknd making the Update procedure more complicated and in line with particleNet. Here each update consists of 3 learnable Dense Layers with included Biases (thus breaking Graph Permutation Symmetry). In between each of these Layers is a Batch Normalisazion Layer and an activation (mlpact)

Arguments

- **gs**: The gs Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the Feature Vector
- **keepconst**: The first keepconst Features are keept unchanced
- **iterations=1**: repeat the Actions of this Layer iterations time
- **alinearity=[-1.0,1.0]**: activation of this Layer, explained better in glm
- **initializer="glorot_uniform"**: Initializer of this Layer
- **i1**: Size after the first Dense Layer
- **i2**: Size after the second Dense Layer
- **mlpact=K.relu**: Activation after each Dense Update Step. Requires to be a function
- **momentum=0.99**: Momentum of the BatchNormalisation
- **k=16**: Number of Average Connections in the Graph. Can be ignored, and is ignored in glm, but migth help the Network converge

3.50 glm

Central and probably most Important Layer of this Package. Updates a Feature Vector using its corresponding Adjacency Matrices and two learnabel Update Matrices. One selfInteraction Matrix that could be understood as a Dense Layer (without bias) acting on each Particle alone, and one neighbour Interaction Matrix, that connects, and acts on, the Node Features in the Way defined in the Adjacency Matrix Inputs an Adjacency matrix and a Feature vector, and returns the updated Feature vector

Arguments

- **gs**: The gs Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the Feature Vector
- **iterations=1**: repeat the Actions of this Layer iterations time
- **alinearity=[-1.0,1.0]**: activation of this Layer, is here written in such a way, that alinearities can be applied even when the Number of iterations is big, since every alinearity is defined in such a way, that appliyng it twice, wont do anything more than appliying it once. This is achieved by using relus to construct a function, that is the Identity between two Values (the two values that are inputtet into alinearity), and the first Value if the Input is below the first Value, as well as it is the second Value, if the Input is bigger than this second Value. To extend this, both Values can be set to minus infinity and infinity respectively, by setting this value to a String. To run other Alinearities, disable this Parameter by setting it to [] and run an Activation Layer afterwards.
- **kernel_initializer="glorot_uniform"**: Initializer of this Layer
- **self_initializer=None**: Instead of using kernel_initializer, this can be used to specify an initializer just for the self interaction of this Layer. Has preference over kernel_initializer.
- **neig_initializer=None**: Instead of using kernel_initializer, this can be used to specify an initializer just for the neighbour interaction of this Layer. Has preference over kernel_initializer
- **learnable=True**: weather this Layer has learnable Variables (self and neighbour interaction). Useful for debugging sometimes

3.51 glom

An early Try of getting a better Update Step, that does not work at the moment, and is only in here for mild technical reasons

3.52 gl

Old Preceding Version of glm, that works on concattet Graphs, but unlike glkeep does not allow for any dimension parameter

Arguments

- **graphmax**: What is usually called gs. The Number of Nodes of the Graph (Graph Size)
- **graphvar**: What is usually called param,The Number of Features for each node in the Feature Vector
- **keepconst**: The Number of Features that are manually kept unchanced by this Layer
- **iterations=1**: repeat the Actions of this Layer iterations time
- **alinearity=[-1.0,1.0]**: activation of this Layer, explained in glm
- **kernel_initializer="glorot_uniform"**: Initializer of this Layer

3.53 gltknd

Precessor of glm. glm works by using a Kronecker Product to convert the Update into only One Matrix. This allows to invert Layers and accelerates high iterations. The central difference to gltknd is that gltknd is not written like this, but calculates each update step on each own. This should mostly be only useful if you require keepconst.

Arguments

- **gs**: The gs Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the Feature Vector
- **keepconst**: The first keepconst Features are kept unchanced
- **iterations**: Repeat the Actions of this Layer iterations time
- **alinearity=[-1.0,1.0]**: activation of this Layer, explained in glm
- **kernel_initializer="glorot_uniform"**: Initializer of this Layer
- **self_initializer=None**: Instead of using kernel_initializer, this can be used to specify an initializer just for the self interaction of this Layer. Has preference over kernel_initializer.
- **neig_initializer=None**: Instead of using kernel_initializer, this can be used to specify an initializer just for the neighbour interaction of this Layer. Has preference over kernel_initializer
- **learnable=True**: weather this Layer has learnable Variables (self and neighbour interaction). Useful for debugging sometimes

3.54 gltk

Precessor of gltknd. Does not allow for different initializer for each Interaction type.

Arguments

- **gs**: The gs Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the Feature Vector
- **keepconst**: The first keepconst Features are kept unchanced
- **iterations**: The Actions of this Layer are repeatet iterations time
- **alinearity=[-1.0,1.0]**: activation of this Layer, explained in glm
- **kernel_initializer="glorot_uniform"**: Initializer of this Layer

3.55 gltrivmlp

Copy of glmlp but with a trivial update procedure (cut to the desired size). Sometimes useful for debugging

Arguments

- **gs**: The gs Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each node in the Feature Vector
- **keepconst**: The first keepconst Features are keept unchanced
- **iterations=1**: repeat the Actions of this Layer iterations time

All remaining Parameters can be given, but have no effect

- **alinearity=[-1.0,1.0]**: activation of this Layer, explained better in glm
- **initializer="glorot_uniform"**: Initializer of this Layer
- **i1**: Size after the first Dense Layer
- **i2**: Size after the second Dense Layer
- **mlpact=K.relu**: Activation after each Dense Update Step. Requires to be a function
- **momentum=0.99**: Momentum of the BatchNormalisation
- **k=16**: Number of Average Connections in the Graph. Can be ignored, and is ignored in glm, but migth help the Network converge

3.56 gmake1graph

Generates a trivial Graph of size (-1,1,1) that is entirely 1 and uses the Batch dimension of the Input

Arguments

none

3.57 gmultiply

Takes a feature Vector and multiplies each Number of Nodes by copiyng it c times. So transforms (-1,gs,param) into (-1,gs*c,param)

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features in each Feature vector
- **c=2**: The Number of repetitions

3.58 gortho

Runs a random, but fixed orthogonal Transformation mixing the Features

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features in each Feature vector
- **seed=None**: seed generating the Transformation

3.59 gpartinorm

Normalises each Feature in each Batch in a special way: After subtracting the mean of each vector x, it subtracts the mean(abs(x)) from it, just to divide it by (mean(abs(x))+max(abs(x)))/2.

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features in each Feature vector
- **alpha=0.01**: A numeric Constant to remove divergences from dividing (instead of 1/a it uses 1/(abs(a)+alpha))

3.60 gperm

Probably a useless Layer since it only works in a 16dimensional Feature Space. Similar to gortho, but uses a (fixed) Permutation Matrix.

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features in each Feature vector

3.61 gpoolgrowth

Takes a List of Featurevectors for one Node (-1,param) and the old Feature vector, to learn from this 2d Vector (out-inn) new nodes (using a 1 layer dense), that are concattet to the old Featurevector and returned

Arguments

- **inn**: The initial Number of Nodes of the Graph (Graph Size)
- **out**: The resulting Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each Node
- **kernel_initializer=glorot_uniform**: The initializer of the Transformation

3.62 gpool

Simple Pooling Layer. Allows you to reduce a 3 dimensional Tensor (-1,gs,param) into (-1,param) by running a Pooling Operation on each Node. Is the simplest way to finish a classical Graph Network that does not break Graph Permutation Symmetry

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each Node
- **mode="max"**: Pooling Operation, either "max","mean" or "sum"

3.63 gpre1

One of the Data Preproccessing Layers that is mostly not useful for anything not Particle Physics related, since it assumes the Input to be a list of lists of Momentum 4 vectors. Is Outdatet, buggy and only here for consistency

Arguments

- **gs**: The Number of Particles used, will become the Number of Nodes (Graph size)
- **numericC=10000**: A Numerical Constant that will be sometimes used to keep things finite

Produced Features

- E
- p1
- p2
- p3
- eta
- phi
- m
- pt (transverse momentum)
- p (absolute Value of the Momentum 3 vector)
- iszero (a flag to filter out missing(zero) particles)

3.64 gpre2

One of the Data Preproccessing Layers that is mostly not useful for anything not Particle Physics related, since it assumes the Input to be a list of lists of Momentum 4 vectors. Is Outdatet, buggy and only here for consistency

Arguments

- **gs**: The Number of Particles used, will become the Number of Nodes (Graph size)
- **numericC=10000**: A Numerical Constant that will be sometimes used to keep things finite

Produced Features

- eta-mean(eta)
- phi-mean(phi)
- ln(pt)
- ln(E)
- -ln(pt/sum(pt))
- -ln(E/sum(E))
- sqrt((eta-mean(eta))**2+(phi-mean(phi))**2)
- iszero (a flag to filter out missing(zero) particles)

3.65 gpre3

One of the Data Preproccessing Layers that is mostly not useful for anything not Particle Physics related, since it assumes the Input to be a list of lists of Momentum 4 vectors. Is Outdatet, buggy and only here for consistency

The only chance to gpre2 is the position of the flag

Arguments

- **gs**: The Number of Particles used, will become the Number of Nodes (Graph size)
- **numericC=10000**: A Numerical Constant that will be sometimes used to keep things finite

Produced Features

- iszero (a flag to filter out missing(zero) particles)
- eta-mean(eta)
- phi-mean(phi)
- ln(pt)
- ln(E)
- -ln(pt/sum(pt))
- -ln(E/sum(E))
- sqrt((eta-mean(eta))**2+(phi-mean(phi))**2)

3.66 gpre4

One of the Data Preproccessing Layers that is mostly not useful for anything not Particle Physics related, since it assumes the Input to be a list of lists of Momentum 4 vectors. Is Outdatet, buggy and only here for consistency

Less Attributes than gpre3

Arguments

- **gs**: The Number of Particles used, will become the Number of Nodes (Graph size)
- **numericC=10000**: A Numerical Constant that will be sometimes used to keep things finite

Produced Features

- iszero (a flag to filter out missing(zero) particles)
- eta-mean(eta)
- phi-mean(phi)

3.67 gpre5

One of the Data Preproccessing Layers that is mostly not useful for anything not Particle Physics related, since it assumes the Input to be a list of lists of Momentum 4 vectors.

Extended and debugged Version of gpre4

Arguments

- **gs**: The Number of Particles used, will become the Number of Nodes (Graph size)
- **numericC=10000**: A Numerical Constant that will be sometimes used to keep things finite

Produced Features

- iszero (a flag to filter out missing(zero) particles)
- eta-mean(eta)
- phi-mean(phi)
- -ln(pt/sum(pt))

3.68 gremoveparam

A simple Layer to remove Features

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **inn**: The initial Number of Features for each Node
- **out**: The resulting Number of Features for each Node

3.69 gshuffle

Shuffles each Featurevector in a random Manner. But compared to gortho, the Transformation is not constant in training but the seed only sets the inital transformation

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features in each Feature vector
- **seed=None**: seed generating the Transformation

3.70 gssort

Sorts a Featurevector in descending Order by its index Feature

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features in each Feature vector
- **index=-1**: Feature Index by which to sort

3.71 gsym

Symmetrises a Adjacency Matrix, by adding the Transposed Matrix to it and rounding it (rounding is simplified by setting the Numerical constant to a low fixed Value of 5). You could understand this as a connection A_ij is one, if either A_ij or A_ji is one.

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)

3.72 gtbuilder

Similar to the other builders on concattet graphs, but uses exactly 2 Adjacency Matrices which are combined in way defined by a constant metrik, creating a concattet graph with only 1 Adjacency Matrix

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)

- **param**: The Number of Features for each Node
- **free**: The Number of empty Features this Node creates

3.73 gtlbuilder

Similar to gtbuilder but the Metrik is learnable and initialised to 1

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each Node
- **free**: The Number of empty Features this Node creates

3.74 gtopk

The probably most useful Graph creation algorithm in this Package. Runs a TopK algorithm, connecting each node to its K neirest neighbours in a Space with a learnable Metrik.

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)
- **param**: The Number of Features for each Node
- **k**: How many connections should each node have
- **free**: How many empty Features does this Layer create
- **learnable**: Weather the metrik should be learnable
- **self_interaction=False**: Should connections from a node to its self be allowed? Please note, that for any metrik with elements below zero does not require that the distance from a node to another node is minimal when both nodes are the same.
- **self_interaction_const=100.0**: To disallow connections between the same nodes, the Layer adds this constant to the distance between those nodes, so this constant should probably be modified if needed and the order of magnitude of the Input is large.
- **metrik_init=keras.initializers.TruncatedNormal(mean=0.0,stddev=0.05)**: Initializer of the metrik defining distances
- **numericalC=10000**: Constant for Numerical Safety
- **emptyconst=100000000.0**: This Layer understands Flags. Sums distances between non flagged Nodes with this constant. It is so much higher than self_interaction_const, since the Graph Permutation Symmetry henges on it
- **flag=7**: The Flag index

3.75 gvaluation

Takes a Featurevector and concats it with a new Feature that is a learnable (simple Dense) Function of the old Features

Arguments

- **gs**: The Number of Nodes of the Graph (Graph Size)

- **param**: The Number of Features for each Node
- **metrik_init="glorot_uniform"**: initializer of the Function
- **learnable=True**: Should the Function be learnable

3.76 multidense

Runs just a list of Dense Layers (defined by m.mdense* and by the parameter q, which gives width and number of Layers) on the last axis of the input data

Arguments

- **g**: A grap Class containing an Adjacency Matrix and a Feature vector, as well as a state Class containing the current standart Values for gs(=Graph Size, Number of Nodes in the Graph) and param (=Parameter Count, Number of Features for each Node)
- **m**: The Constants defining the Function Behaviours (matching Parameters below)
- **q**: An Array setting the size of each Layer [8,4,7] creates 3 Dense Layers with 8,4 and 7 nodes respectively

Constants defined in m

- **mdense_activation="relu"**: The Activation of each Dense Layer
- **mdense_init_kernel=tf.keras.initializers.Identity()**: Kernel Initializer of each Layer
- **mdense_init_bias=tf.keras.initializers.Zeros()**: Bias Initializer of each Layer
- **mdense_usebias=True**: Should this Networks use a Bias Term
- **mdense_batchnorm=False**: Should you use a BatchNormalisationLayer after each Layer

3.77 norm

Normalises a network on the last axis, scale decides if there is a learnable multiplicative factor

Arguments

- **g**: A grap Class containing an Adjacency Matrix and a Feature vector, as well as a state Class containing the current standart Values for gs(=Graph Size, Number of Nodes in the Graph) and param (=Parameter Count, Number of Features for each Node)
- **scale=True**: Weather the output of the BatchNormalisationLayer has a learnable scaling Factor. Can be useful to disable this in special Situations, for example, when working in front of an Autoencoder

3.78 prep

Runs my standart preparation on an Input which it defines itself and also returns this Input

Arguments

- **g**: A grap Class containing an Adjacency Matrix and a Feature vector, as well as a state Class containing the current standart Values for gs(=Graph Size, Number of Nodes in the Graph) and param (=Parameter Count, Number of Features for each Node)
- **m**: The Constants defining the Function Behaviours (matching Parameters below)

Constants defined in m

- **prenorm=False**: Should each Feature be normalised (using norm with scale=False) after preperation

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

CHAPTER 5

Functional Api

The Functional Api is less powerful than the Layer Api, but also easier to handle. And since some functions are really complicated, using the Functional Api as much as possible is usually recommended.

Contents:

5.1 multidense

A function to add multiple Dense Layers with parameters defined by constants like m.dense*, aswell as the node numbers of the values of the list q. Dense Layers only update the last axis of a Tensor.

Arguments:

- **g**: a grap object
- **m**: a constant object (generated for example by getm())
- **q**: a list of node numbers

Returns:

- **g**: the updated grap object

5.2 norm

Normalises a network on the last axis (using keras BatchNormalization layer), scale decides if there is a learnable multiplicative factor

Arguments:

- **g**: a grap object
- **scale=True**: should the BatchNormalization layer include a scaling factor (disable if infront of your autoencoder)

Returns:

- **g**: the updated grap object
- **inp**: the model input layer

5.3 prep

Runs my standart Input preparation. Probably not useful except for physics (create an Input, gpre5 on it and optionally use norm (decided by m.prenorm))

Arguments:

- **g**: a grap object, already containing gs and param in the state object s
- **m**: a constant object (generated for example by getm())

5.4 gq

function to work with alternative Input format (here Dense Layer on concat(self_values, neigbour_values)), migth be extended to use Convolutions. Defined by m.gq*

Arguments:

- **g**: a grap object
- **m**: a constant object (generated for example by getm())
- **steps=4**: how many update steps between after glcreate

Returns:

- **g**: the updated grap object

5.5 gaq

like gq but to work on a bit more abstract data (defined by m.gaq*) (for use in graphs of graphs)

Arguments:

- **g**: a grap object
- **m**: a constant object (generated for example by getm())
- **a**: which abstraction constant is used
- **steps=4**: how many update steps between after glcreate

Returns:

- **g**: the updated grap object

5.6 gnl

a function to just add some graph update functionality without relearning the graph, defined by m.graph*. Can use usei to use inverted Graph update layers instead of the normal ones (to make invertibility easier). Also understands alin (iarities) as a vector

Arguments:

- **g**: a grap object
- **m**: a constant object (generated for example by getm())
- **alin=[]**: alinearity used, defined in glm
- **iterations=1**: run each graph update step iteration times (one layer)
- **repeat=1**: repeat this function repeat times (multiple layers)
- **usei=False**: use inverse graph update layers

Returns:

- **g**: the updated grap object

5.7 learngraph

Learns a graph (g.A) as a function of the parameters (g.X). Can also add new parameters to g.X (with free) and you can specify how many connections each node should have (k), mainly used by gll

Arguments:

- **g**: a grap object
- **free=0**: add how many new free parameters to each feature vector
- **k=4**: the k used in the topK layer

Returns:

- **g**: the updated grap object

5.8 gll

gnl + learngraph

Arguments:

- **g**: a grap object
- **m**: a constant object (generated for example by getm())
- **free**: add how many free parameters to each feature vector
- **alin=[]**: alinearity used, defined in glm
- **iterations=1**: run each graph update step iteration times (one layer)
- **repeat=1**: repeat this function repeat times (multiple learnings)
- **subrepeat=1**: repeat the gnl function this many times (multiple layers)
- **usei=False**: use inverse graph update layers

- **k=4**: the k in the topK algorithm

5.9 ganl

gnl but on more abstract graphs, should probably not be used directly unless you unstand what the difference is

Arguments:

- **A**: an Adjacency Matrix
- **X**: a list of Feature vectors
- **gs**: the node number
- **a**: the abstraction factor
- **param**: how many parameters for each feature vector
- **iterations=1**: run each graph update step iteration times (one layer)
- **alin=[]**: alinearity used, defined in glm
- **usei=False**: use inverse graph update layers

Returns:

- **X**: the updated feature object

5.10 abstr

uses (multiglam) glam to abstract a graph into a factor c smaller graph

uses pooling to go from c size subgraphs to 1 size dots

does not chance param at all

uses (pmode) param pooling mode uses (gmode) graph pooling mode

Arguments:

- **g**: a grap object
- **c**: the abstraction factor
- **alin=[]**: alinearity used, defined in glm
- **iterations=1**: run each abstracted graph update step iteration times (one layer)
- **repeat=1**: repeat this function repeat times (multiple abstractions)
- **multiglam=1**: repeat the graph updatedd function this many times (multiple layers)
- **pmode="max"**: how to merge feature vectors. Options defined in gcompoolmerge
- **gmode="mean": how to merge the adjacency matrix. Options defined in gcomgraphcutter

Returns:

- **g**: the updated grap object

5.11 compress

little brother of abstr, the main difference is, that this does not keep any information of the graph, so you have to retrain it, if you want to do graph actions afterwards

Arguments:

- **g**: a grap object
- **m**: a constant object (generate for example by getm())
- **c**: the abstraction factor
- **addparam**: add how many new parameters

Returns:

- **g**: the updated grap object

5.12 graphatbottleneck

handles the bottleneck transformations for a pure graph ae, return g, compressed, new input, shallfp=True=>convert vector in matrix (with gfromparam), can use redense to add a couple dense layers around the bottleneck (defined by m.redense*)

Arguments:

- **g**: a grap object
- **m**: a constant object (generate for example by getm())
- **shallfp=True**: reshapes the 2 dimensional vector (-1, latent_size) into a 3 dimensional vector (-1,g.s.gs,g.s.param) after this function only if this is true

Returns:

- **g**: the updated grap object

5.13 denseladder

helper function that generates a list of Dense sizes going from 1 to c in n steps (excluding 1 and c), c can be a list, than returns a list of lists

Arguments:

- **c**: how many nodes should be the final node number. if is a list, repeats this layer for each value and returns then a list of lists
- **n=3**: how many steps to take
- **truestart=False**: start with 1?

Returns:

- **l**: a list of integers

5.14 divtriv

trivial graph diverger by a factor of c (does not chance param at all)

requiregp=True: require ggoparam at the start addDense: intermediate Dense Layers, sizes between 1 and c useful

Best handled decompress

Arguments:

- **g**: a grap object
- **c**: the abstraction factor
- **m**: a constant object (generate for example by getm())
- **shallgp**: if the input is 3 dimensional, set this to true. if it is already 2 dimensional (since graphatbottleneck) set it to false
- **addDense=[[]]**: intermediate Dense layer sizes
- **activation**: activation of the dense layers

Returns:

- **g**: the updated grap object

5.15 divccll

easy diverger: diverge by copy Best handled by decompress

Arguments:

- **g**: a grap object
- **c**: the abstraction factor

Returns:

- **g**: the updated grap object

5.16 divpar

A parameter like graph diverger by a factor of c (also does not chance param at all) Best handled by decompress

Arguments:

- **g**: a grap object
- **c**: the abstraction factor
- **usei=False**: Use inverse graph update steps
- **alin=[]**: alinearity of the graph update steps, defined in glm
- **iterations=1**: repeat each graph update step this many time (one layer)
- **repeat=1**: repeat this layer repeat time (multiple divergences)
- **multiglam=1**: multiglam graph update steps (multiple layers)

- **amode2=”prod”**: combine graphs using this function, options defined in gcomgraphand2

Returns:

- **g**: the updated grap object

5.17 divcla

classic graph abstractor, also does not chance the paramsize, just goes from one param to c params, and has one learnable matrix (which is const between the elements). Works by usual parameter divergence, and then by abstracting the graphs, with the constant learnable one Best handled by decompress

Arguments:

- **g**: a grap object
- **c**: the abstraction factor
- **m**: a constant, defined for example by getm()
- **repeat=1**: repeat this layer repeat time (multiple divergences)

Returns:

- **g**: the updated grap object

5.18 divcla2

even more simple divcla, the main difference is, that this ignores graphs completely Best handled by decompress

Arguments:

- **g**: a grap object
- **c**: the abstraction factor
- **m**: a constant, defined for example by getm()
- **repeat=1**: repeat this layer repeat time (multiple divergences)

Returns:

- **g**: the updated grap object

5.19 divgra

graph like graph diverger by a factor of c (also does not chance param at all)

amode [and operation modus for graphand] amode2 : and operation modus for graphand2

Best handled by decompress

Arguments:

- **g**: a grap object
- **c**: the abstraction factor
- **m**: a constant variable. created for example by getm()

- **usei=False**: Use inverse graph update steps
- **alin=[]**: alinearity of the graph update steps, defined in glm
- **iterations=1**: repeat each graph update step this many time (one layer)
- **repeat=1**: repeat this layer repeat time (multiple divergences)
- **multiglam=1**: multiglam graph update steps (multiple layers)
- **amode="prod"**: combine parameters by this, options defined in gcomgraphand
- **amode2="prod"**: combine graphs using this function, options defined in gcomgraphand2

Returns:

- **g**: the updated grap object

5.20 remparam

just a simple function to remove overdue parameters

Arguments:

- **g**: a grap object
- **nparam**: output parameter number

Returns:

- **g**: the updated grap object

5.21 handlereturn

a nice function to simplify returning values for createbothmodels. Also has some simple size consistency checks
the variables:

Arguments:

- **inn1**: initial input Variable
- **raw**: preconverted input Variable, for comparison sake
- **com**: compressed Variable
- **inn2**: input for decoder
- **decom**: decompressed decoder Variable
- **shallvae**: shall you thread this like a variational auto encoder? hier just a bodge of an solution

Returns:

- **inn1**: the first input
- **raw**: preprocessed value
- **c1**: mean/latent space
- **c2**: variance/latent space
- **shortcuts=[]**: shortcut variable, disabled here
- **inn2**: the decompression input

- **decom**: output value

5.22 sortparam

sorts X by one of its parameters (m.sortindex), just removes the graph

Arguments:

- **g**: a grap object
- **m**: constant variable, created for example by getm()

Returns

- **g**: the updated grap value

5.23 subedge

one particlenet like update step, uses m.edge*

Arguments:

- **inp**: input variable
- **param**: number of parameters
- **m**: constant variable, created for example by getm()

Returns

- **feat3**: an updated feature vector

5.24 edgeconv

one set of particlenet like update steps, thus use m.edge* like subedge. also similar to gq (here the main difference is the dense vs convolutional structure

Arguments:

- **inp**: input variable
- **gs**: node number
- **k**: the k in topK
- **param**: number of parameters
- **m**: constant variable, created for example by getm()

Returns

- **outp**: the updated feature vector

5.25 ge

the upper level managing particlenet like update steps (like edgeconv and subedge, can mostly use m.edgeconcat to decide if you should concat or replace the output (concat:like particlenet, replace:probably better for autoencoder)

Arguments:

- **g**: a grap object
- **m**: constant variable, created for example by getm()
- **k=4**: the k in topK

Returns

- **g**: the updated grap value

5.26 shuffleinp

shuffles the inputs, cross particle. . . sadly does not keep the shuffle constant

Arguments:

- **g**: a grap object
- **seed=None**: optional seed

Returns

- **g**: the updated grap value

5.27 orthoinp

like shuffleinp, but uses an orthogonal matrix instead of shuffle, thus constant, but mixes the inputs in a certain way

Arguments:

- **g**: a grap object
- **seed=None**: optional seed

Returns

- **g**: the updated grap value

5.28 perminp

like orthoinp, but uses an permutation matrix instead of an orthogonal one. migth require some improvements in gperm.py before it becomes truly useful

Arguments:

- **g**: a grap object

Returns

- **g**: the updated grap value

5.29 pnorm

runs a normation on each particle and feature, ignoring the first one ignores the first variable. and the normation is defined in gpartinorm

Arguments:

- **g**: a grap object

Returns

- **g**: the updated grap value

5.30 prevcut

cuts in gs, takes only the last ops values

Arguments:

- **g**: a grap object
- **ops=4**: returns only the last ops nodes

Returns

- **g**: the updated grap value

5.31 goparam

transforms the 3d (-1,gs,param) data into 2d (-1,gs*param) ones. You can use chanceS to disallow this function to chance the settings

Arguments:

- **g**: a grap object
- **chanceS=True**: chances the variable of g.s is this is True

Returns

- **g**: the updated grap value

5.32 decompress

function to run diverge algorithms on the input. You can choose the diverge algorithm with m.decompress (trivial, paramlike,graphlike,classic,classiclg,ccll), c can be a list (multiple divergences) and also handles the bottleneck actions (define a new input, and return it later). Always returns: g,compressed version,new input

Arguments:

- **g**: a grap object
- **m**: a constant variable. created for example by getm()
- **c**: the abstraction factor

Returns

- **g**: the decompressed grap value
- **com**: latent space vector
- **inn2**: input for the decompressor

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

CHAPTER 7

Indices and tables

- genindex
- modindex
- search