Deep neural networks (DNNs)

209

Deep neural networks (DNNs)

Definition ver 1.0

- Just networks with many layers

Up to 14 year ago, researchers rarely used deep networks

- at that time, no theoretical advantage of DNNs
  - Deep networks are universal approximators:
    - Obvious proof: two layers can approximates any function, the remaining layers can implement the identity function
  - No clear advantage in generalization result for deep networks
- Experimental results (on simple application) had not shown any advantage from DNNs
- DNNs suffered from long-term dependence problem!!

210

Deep neural networks

Then researchers started to suspect that

- Very difficult problems may require the use of several layers: computer vision, language understanding,…
- New learning methods may have to be used to overcome the long-term dependence problem:
  - Unsupervised learning for hidden layer?
  - Stacked supervised learning?
  - A lot of parameters ….

Intuitive evidences

- Each layer produces a more abstract representation of inputs simplifying difficult problems in an iterative way
- Animal brains are layered … , it is not a casuality

211

Current DNNs are much more complex than old DNNs

Current DNNs exploits a lot of peculiarities

- Different types of layers
- Weight sharing
  - Some neurons share the same weights
- Modularization
  - A sub-module of the network is applied on different subset of the inputs
- particular activation functions
  - Rectifier, drop out, max out, …

212

An example of DNNs: convolutional neural networks

Convolutional neural networks

- For image classification
- Originally used for handwritten digit recognition
- Currently, the best tools for object recognition sin images are based on evolutions of convolutional neural networks

The layers of a convolutional network

- Convolutional layers
- Pooling layers
- Fully connected layers

213

Convolutional neural networks: convolutional layer

Kernel filters in image processing

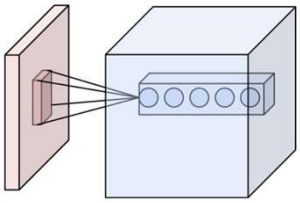
- Filter kernels are matrices, which can be applied to an image by convolution

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$ Edge Detection $$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$ Blur

- By convolution with a 3x3 matrix M, the pixels of each 3x3 window are multiplied by M
- A convolutional layer is almost equivalent to the application of a kernel whose parameters are trained!!

214

Convolutional neural networks: convolutional layer

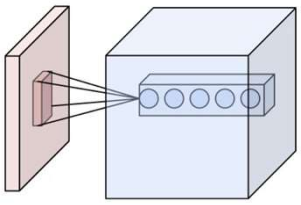


Convolutional layers

- A neuron of a convolutional layer implements a kernel
  - The neuron is connected to a window (receptive field) of the original image (f.i., a 3x3(x3) square)
  - The kernel matrix is defined by the weights of the connections
- The kernel is convoluted over the input
  - There is a neuron for each window receptive field
  - All the neurons share the same weight sets

215

Convolutional neural networks: convolutional layer



- A convolutional layer has the dimension of the input image (width and height)
- Intuitively, convolutional layers extract low level features from the input image

216

Convolutional neural networks: pooling layer

Pooling layers

- Each neuron of a pooling layer summarize the result of a small window of the image
  (f.i., a 2x2 receptive fields in the previous layer)
- Summarization can be by taking maximum, a random value, …
- Pooling layers decrease the size of the features (decrease width and height)

- Intuitively, pooling layers are used to simplify the problem by reducing the features to be considered

217

Convolutional neural networks: fully connected layer

Fully connected layers

- Common layers in which each neurons connected to all the neurons of the previous layer

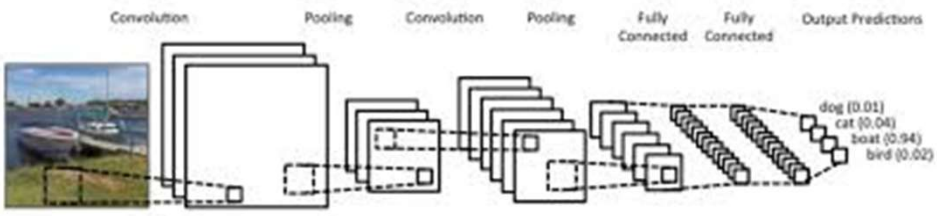
- Intuitively, fully connected layers allow to combine and reason about the extracted features in order to take the final decision

218

Convolutional neural networks

The architecture

- A convolution layer followed by a pooling layer
- One or several fully connected layer
- The pair (convolutional, pooling) may be repeated several times
- Several convolutional layers can be used in parallel



219

Current DNNs

A lot of different architectures are now available

- 1998, LeNet 6 layers, 68K parameters
- 2012, Alexnet 8 layers, 60M parameters
- 2014, Inception, 22 layers, 2M parameters
- 2014, VCGG net, 16 layers, 138M parameters
- 2015, ResNet 152 layer, 60M parameters

Same image datasets

- Imagenet 14M images, 1000 categories
- CIFAR 100, 60K images, 100 categories
- MNIST 60K handwritten characters
- ….

Current convolutional architettures are over-parametrized!!!

- More parameters than images
- What are the consequences of this?

220

UNIVERSITÀ DI SIENA 1240

Current DNNs

Current convolutional architettures are over-parametrized!!!

- Learning is simple, not much affected by local minima
  - this is expected
- Generalization may be also good
  - this is achieved only applying a number of tricks
    - Pooling
    - Weight decay
    - Use validation to stop
    - Dataset aumentantion
      - Images can be rotated, scaled, ….

221

UNIVERSITÀ DI SIENA 1240

Current DNNs

An experiment Zang, et al.

- with CIFAR 10 dataset (60K images, 10 classes)
- Using Inception (1.6M par)
- What does it happen without the usual "tricks"?
  - The networks easily reaches 100% on train set,
  - but of course, the generalization is low
- What does it happen if the labels are replaced by random labels? And what if also the image pixels are shuffled, randomly permuted…
  - The networks easily still reaches 100% on train set,
  - but of course, now the performance on test set is that of a random classifier

Let us have a look at

https://arxiv.org/abs/1611.03530

222

Approximation capability of DNNs

Approximation capability ver 1.0

- DNNs are universal approximators (with mild constraints on architecture)
- An example of the proof,
  - Given a target function t,
  - the first three layers of the DNN approximate t
  - remaining layers just copy the ouptut of the third layer

223

The role of depth in DNNs

Bianchini, Scarselli

Approximation capability ver 2.0: the idea

"(using the same of amount of resources), deep architectures can implement more complex functions than shallow networks"

A new tool was required to evaluate the complexity of the implemented classifiers

- The following complexity measures were not useful
  - Number of neurons
  - VC-dimension
  - Approximation capability

224

The underlining idea

- *N*: a neural network
- *f*: the function implemented by the network

- S_N: the set of the non-negative patterns, i.e.
  $S_N=\{x|\ f_N(x)\geq 0\}$

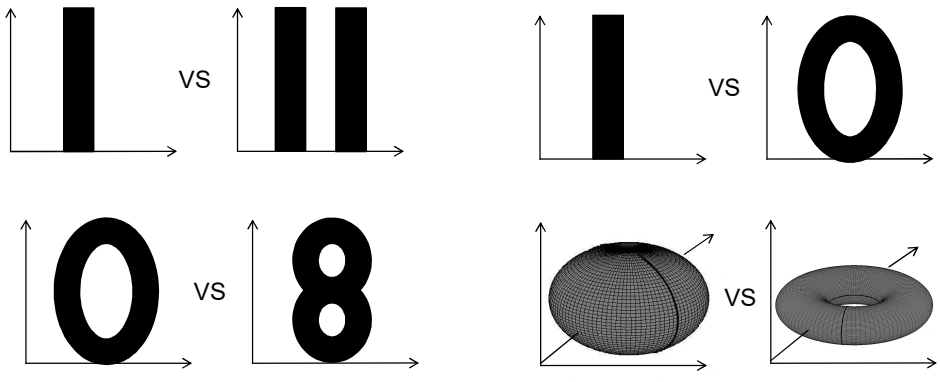
The idea

- To measure the topological complexity of the set S_N
- It is reasonable when *N* is used for classification purposes

225

Topological complexity: an intuitive viewpoint

- Black regions represent the non-negative patterns, i.e,, S_N
- Can you say which set is more complex in each couple?



226

Betti numbers: a topological concept

Betti numbers

- used to distinguish spaces with different topological properties
- for any subset $S \subset R^n$, there exist n Betti numbers,
  $b_0(S), b_1(S), \ldots. b_{n-1}(S)$

Formally

- $b_k(S)$: is rank of the k–th homology group of the space S.

Intuitively

- $b_0(S)$: is the number of connected components of the set S
- $b_k(S)$: counts the number of $(k+1)$–dimensional holes in S

227

The proposed measure of complexity

- In topology, the sum of the Betti numbers

  $B(S)=\Sigma_k\, b_k(S),$

  is used to evaluate the complexity of the set S

- We will measure the complexity of the function f_N implemented by a neural network N by

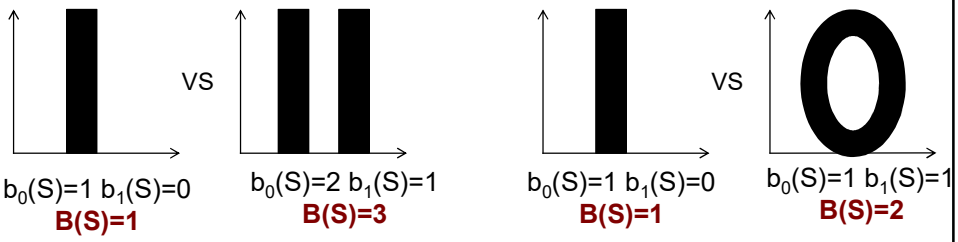
  $B(S_N)=\Sigma_k\, b_k(S_N),$

  where of $S_N=\{x|\ f_N(x)\geq 0\}$.

228

UNIVERSITÀ DI SIENA 1240

Betti numbers: examples

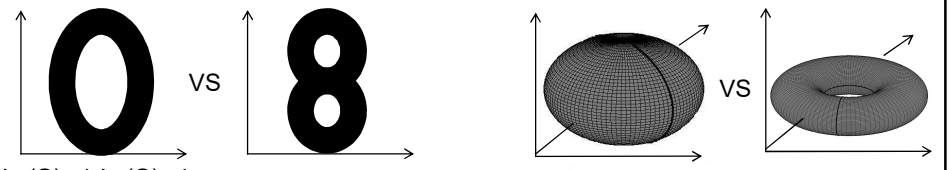


$b_0(S)=1\ b_1(S)=0$ **B(S)=1**

$b_0(S)=2\ b_1(S)=1$ **B(S)=3**

$b_0(S)=1\ b_1(S)=0$ **B(S)=1**

$b_0(S)=1\ b_1(S)=1$ **B(S)=2**



$b_0(S)=1\ b_1(S)=1$ **B(S)=2**

$b_0(S)=1\ b_1(S)=2$ **B(S)=3**

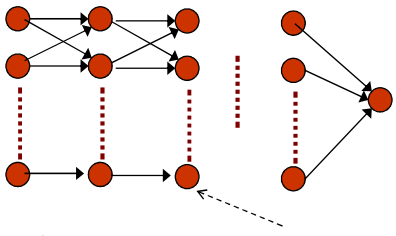
$b_0(S)=1\ b_1(S)=1$ $b_2(S)=1$ **B(S)=3**

$b_0(S)=1\ b_1(S)=2$ $b_0(S)=1$ **B(S)=4**

229

UNIVERSITÀ DI SIENA 1240

The study



The considered networks

- feedforwad layered perceptrons
- with sigmoidal (ridge) activation functions in the hidden layers and linear in the output layer

Upper and lower bounds on $B(S_N)$ *varying*

- The number of hidden layers *l*
- The number of hidden units *h*
- The number of inputs *n*
- The activation functions: tanh, arctan, polynmial of degree *r*, generic sigmoids

230

UNIVERSITÀ DI SIENA 1240

The results

- *l*: number of hidden layers
- *h*: number of hidden units
- *n:* number of inputs
- *r:* degree of the polynomial

Layers	Activation	Bound on $B(S_N)$	Exponential	Polynomial
Upper bounds				
1	threshold	h^n	n	h
1	polynomial	$(2 + r)(1 + r)^{n-1}$	n	r
1	arctan	$(n + h)^{n+2}$	n	h
many	arctan	$2^{h(2h-1)} \bullet (nl + n)^{n+2h}$	n,h,l	
many	tanh	$2^{h(h-1)/2} \bullet (nl + n)^{n+h}$	n,h,l	
many	polynomial	$(2 + r^l)(1 + r\ l)^{n-1}$	n, l, h	r
Lower bounds				
1	sigmoid	$((h-1) /n)^n$	n	h
many	sigmoid	2^{l-1}	l, h	
many	polynomial	2^{l-1}	l, h	

231

UNIVERSITÀ DI SIENA 1240

Analysis of the results

W.r.t. the number of inputs n, the complexity

- always grows exponentially

232

UNIVERSITÀ DI SIENA 1240

The results

- *l*: number of hidden layers
- *h*: number of hidden units
- *n:* number of inputs
- *r:* degree of the polynomial

Layers	Activation	Bound on $B(S_N)$	Exponential	Polynomial
Upper bounds				
1	threshold	h^n	**n**	*h*
1	polynomial	$(2 + r)(1 + r)^{n-1}$	**n**	*r*
1	arctan	$(n + h)^{n+2}$	**n**	*h*
many	arctan	$2^{h(2h-1)} \bullet (nl + n)^{n+2h}$	**n**,*h*,*l*	
many	tanh	$2^{h(h-1)/2} \bullet (nl + n)^{n+h}$	**n**,*h*,*l*	
many	polynomial	$(2 + r^l)(1 + r\,l)^{n-1}$	**n**, *l*, *h*	*r*
Lower bounds				
1	sigmoid	$((h-1) /n)^n$	**n**	*h*
many	sigmoid	2^{l-1}	*l*, *h*	
many	polynomial	2^{l-1}	*l*, *h*	

233

UNIVERSITÀ DI SIENA 1240

Analysis of the results

w.r.t. the number of hidden neurons h, the complexity

- grows polynomially, for shallow networks
- grows exponentially, for deep networks

234

The results

- *l*: number of hidden layers
- *h*: number of hidden units
- *n:* number of inputs
- *r:* degree of the polynomial

Layers	Activation	Bound on $B(S_N)$	Exponential	Polynomial
Upper bounds				
1	threshold	h^n	n	**h**
1	polynomial	$(2 + r)(1 + r)^{n-1}$	n	r
1	arctan	$(n + h)^{n+2}$	n	**h**
many	arctan	$2^{h(2h-1)} \bullet (nl + n)^{n+2h}$	n,**h**,l	
many	tanh	$2^{h(h-1)/2} \bullet (nl + n)^{n+h}$	n,**h**,l	
many	polynomial	$(2 + r^l)(1 + r\,l)^{n-1}$	n, l, **h**	r
Lower bounds				
1	sigmoid	$((h-1) /n)^n$	n	**h**
many	sigmoid	2^{l-1}	l,**h**	
many	polynomial	2^{l-1}	l,**h**	

235

Analysis of the results

Summing up

- with the same amount of resources, deep networks can realize more complex classifiers than shallow networks!!

Remarks

- *This does mean that all the functions can be better approximated by deep networks!!*
- *Only some functions with particular symmetries will have benefict from being approximated by deep networks*

236

An intuitive explanation of the advantage of deep architectures

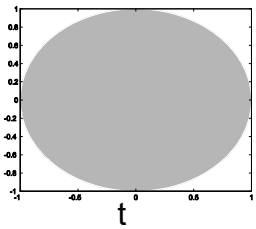
Notice that

- A layered network implements a function
  $f_N = g_1 \circ g_2 \circ .. \circ g_l \circ t$
  - g_k is the function implemented by layer k
  - $\circ$ is the function composition operator
- If $f_N = g_1 \circ t$, then f_N behaves as t on all the regions $A_{1,..}, A_s$ where $g_1(A_k) = R^n$
- With several layers, the number of regions A_k such that $g_l(\ldots g_1(A_k)) = R^n$ can grow exponentially

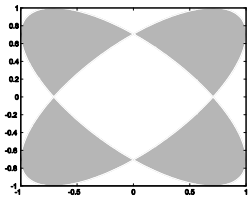
Deep networks can replicate more easily the same behavior on different regions of the input domain

237

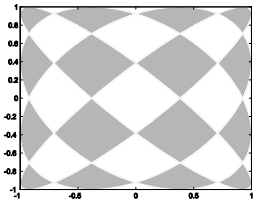
An example of the advantages of deep networks



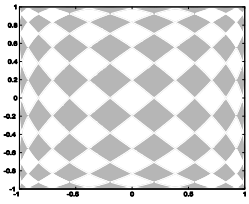
t



g ◦ t



g ◦ g ◦ t



g ◦ g ◦ g ◦ t

$$g(x) = 1 - \|x\|^2$$

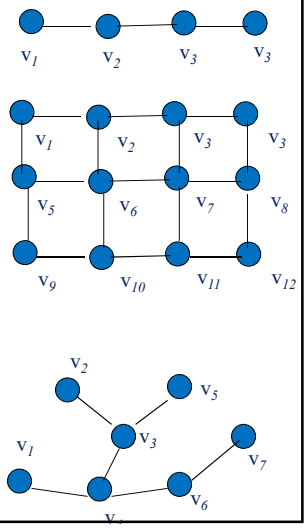
$$t(x) = \left[1 - x_1^2, 1 - x_2^2\right]$$

238

Graph Neural Networks (GNNs)

The main idea

- graphs are an extension of sequences
  - Recurrent networks operate on sequences of data
- graphs are irregular grids
  - Convolutional networks operate on grids of data
- GNNs a class neural network models that extend both recurrent networks and convolutional networks and operates on graphs



239

Graphs: they are a general tool to represent data

Patterns with their relationships, examples of applications for GNNs

- Social networks
  - Nodes represent users (may have attached information)
  - Links represent friendships (may have attached information)
- Protein networks
  - Node stand for proteins (may have attached information)
  - Edges denote their possible interaction (may have attached information)
- Molecules
  - Nodes stand for atoms (may have attached information)
  - Edges stand for physically close atoms or those atoms having an atomic interaction (may have attached information)
  - ….

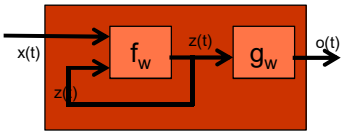
240

Graph Neural Networks (GNNs)

- The first model to be named GNN has been proposed here in Siena, but that before that there were a lot of works on neural networks for graphs, particularly in Pisa, Florence and Siena (e.g. Sperduti, Frasconi, we in Siena,….) but also elsewhere in Europe
- Currently, several new GNN models have been proposed.
- In the following, first I introduce the original model and then I explain how this has been modified

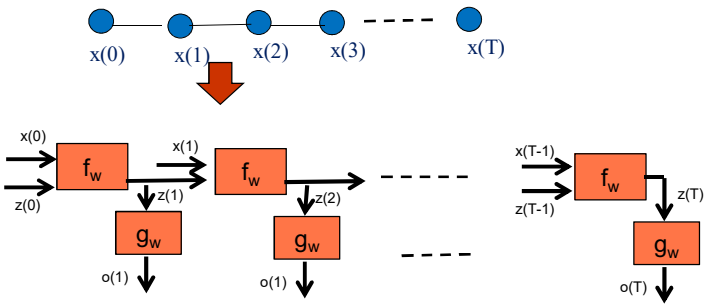
241

GNNs can be defined by extending recurrent networks



Remember the unfolding network

- I recurrent networks, we have a transaction network f_w and an output network g_w
- A copy of f_w, g_w for each time instance, connected in sequence

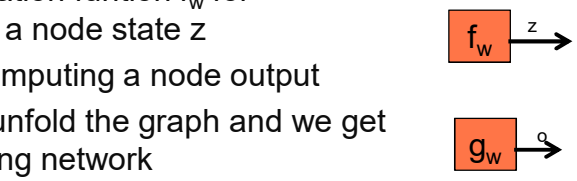


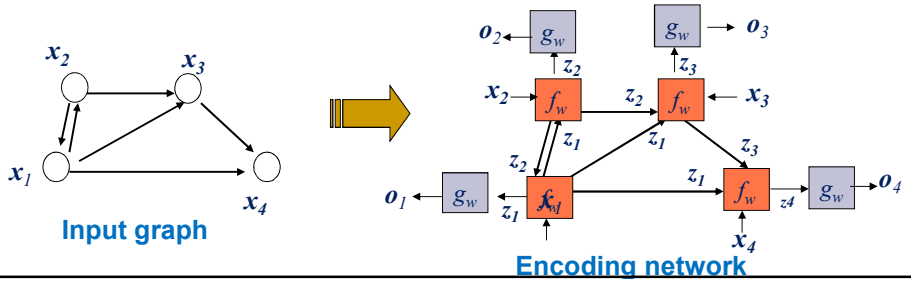
242

GNNs can be defined by extending recurrent networks

We still have two neural networks

- An aggregation funtion f_w for computing a node state z
- A g_w for computing a node output
- Then, we unfold the graph and we get the encoding network



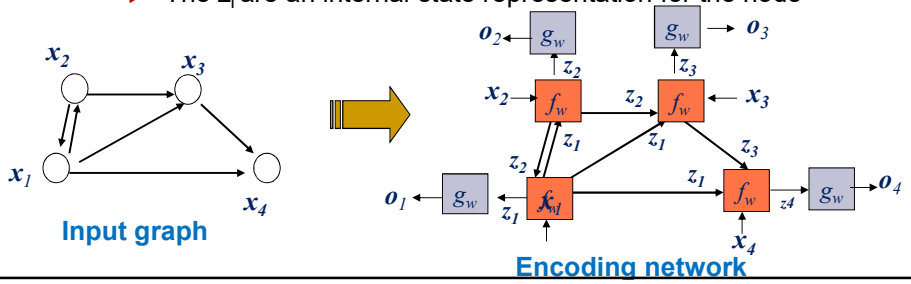


243

GNNs can be defined by extending recurrent networks

For a general graph, we can construct the encoding network

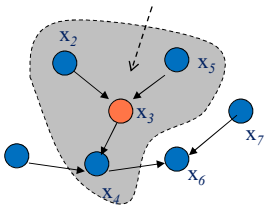
- The x_i are the feature vectors attached to nodes (the node information)
- The edges represent the relationship between patterns
- The o_i are the network output
- The z_i are an internal state representation for the node



244

GNNs: computing the states

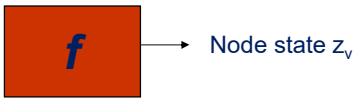
The neighborhood of this node



- The state z_n of a node are computed by combining the information on node neigborhood
- In the original model, we used

$$z_v = \sum_{n \to v} h_w(x_n, z_n)$$

where h_w is a three layer neural network



- but, several other networks are now used ..

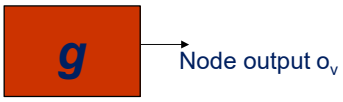
245

GNNs: computing the outptut

- The state o_n of a node are computed from the states z_n
- In the original model, we used

$$z_v = g_w(x_v, z_v)$$

where g_w is a three layer neural network



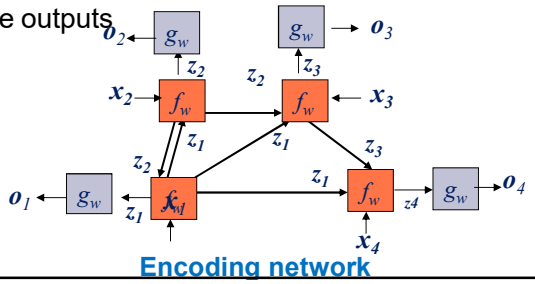
- but, several other networks are now used and sometime is just not used…. back on this later

246

Computing GNN outputs

The enconding network is cyclic … how are outputs computed?

1. Set initial states z(t)=0
2. **Repeat**
3. activate units *f* to compute new states z(t+1)
4. **Until** z(t) do not change any more
5. activate units g to compute outputs



Encoding network

247

Computing GNN outputs

The enconding network is cyclic …

- Does the forward phase always converge?
- Does the forward phase always converge to the same state despite the initial state?

Yes

- The original GNNs adopt a mechanism based on contraction systems

248

A mathematical point of view

Consider the whole encoding network as adynamic system/system of equation does the system always converge/ does the system has a unique solution?

$$Z(t) = F(X, Z(t))$$

- Yes, provided that it is a contraction, i.e, if the norm of the Jacobian is smaller than
$$\left\| \frac{\partial F(l, Z)}{\partial Z} \right\| < 1$$
- The original GNNs adopt a mechanism which

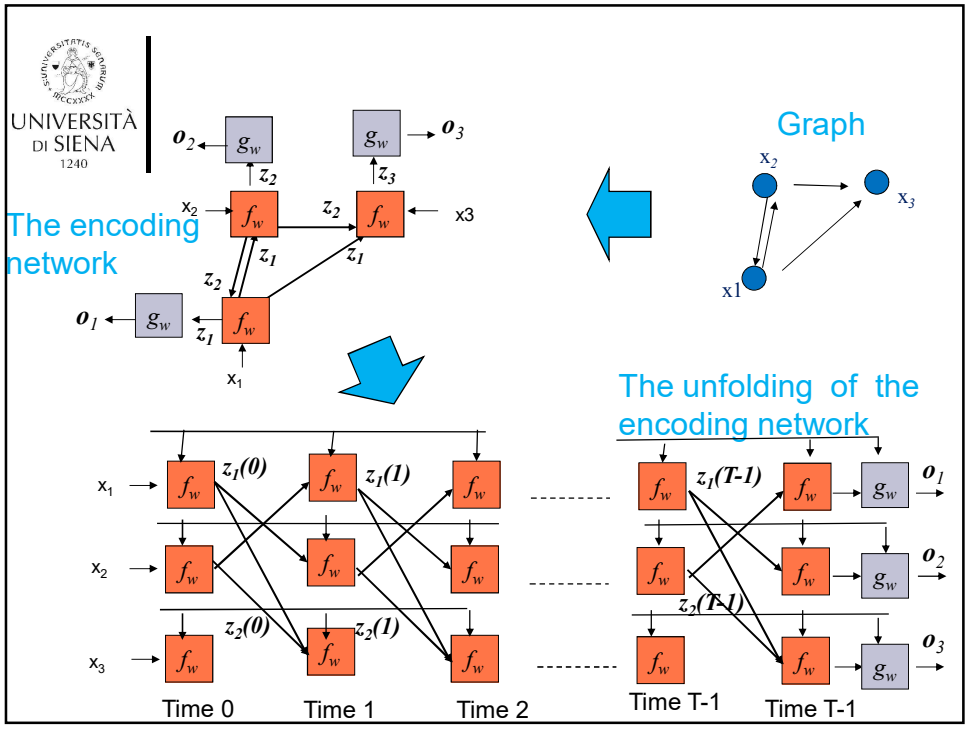
249

Computing GNN gradient

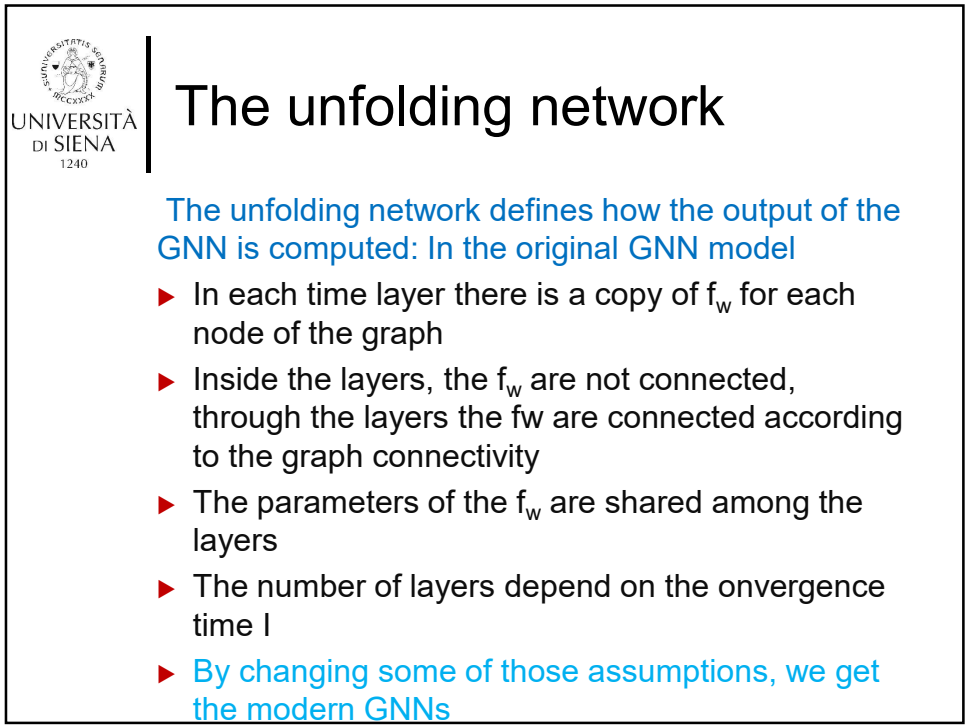
Based on unfolding of encoding network

- The encoding network is unfolded through time
  - The result is a feedforward network equivalent to the encoding network
- The gradient is computed on the unfolding using backpropagation through time algorithm

250



251



The unfolding network

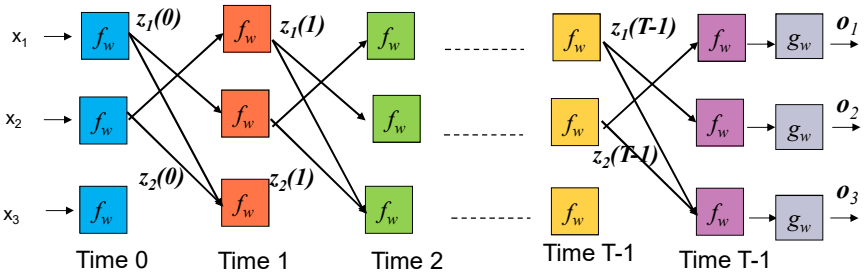
The unfolding network defines how the output of the GNN is computed: In the original GNN model

- In each time layer there is a copy of f_w for each node of the graph
- Inside the layers, the f_w are not connected, through the layers the fw are connected according to the graph connectivity
- The parameters of the f_w are shared among the layers
- The number of layers depend on the onvergence time I
- By changing some of those assumptions, we get the modern GNNs

252

Modern GNNs

- Fix the number of layers
- Use different parameter set on each layer
- You may also want to remove the direct input to the internal layers
- Now, this looks like a deep network



253

Modern GNN: Graph convolutional neural networks (GCN)

- Graph convolutional neural networks is a well know example of modern GNNs
- In GCNs, the aggregation function is

$$z_v(k) = \sum_{n \to v} relu\left(\frac{z_v(k-1)\, W_k}{|ne[v]||ne[n]|}\right)$$

Kipft, Welling

254

Are GNNs sort of convolutional neural networks?

- In convolutional networks, a kernel is applied to each point v of an image (a grid) or of the previous layer
- The oputput of the kernel is

$$z_n = \sum_{v\ in\ ne[n]} relu(x_v W_{n\_v})$$

- In GNN a function f_w is used to combine the contribution of the neigboors

$$z_v = \sum_{n \to v} h_w(x_n, z_n)$$

255

Are GNNs sort of convolutional neural networks?

- In convolutional neural networks, the kernel is applied on each pixel of the input image
  - In GNNs, the aggregation function is applied on each node.
- In convolutional neural networks, several kernels are used to obtained several feature map on each layer
  - In GNNs, the aggregation function produced a state whose dimension is larger than one: you can think this as a set of aggregation functions

256

About GNN computational capability

- Which applications on graph can GNNs implement?
- Let us consider a function φ that takes in input a graph G and return a real output $\varphi(G,n)$ for some node n
- Answer 1.0

The orignal GNN model, under mild assumptions on $\varphi(G,n)$, can approximate, any function φ in probability

257

About GNN computational capability

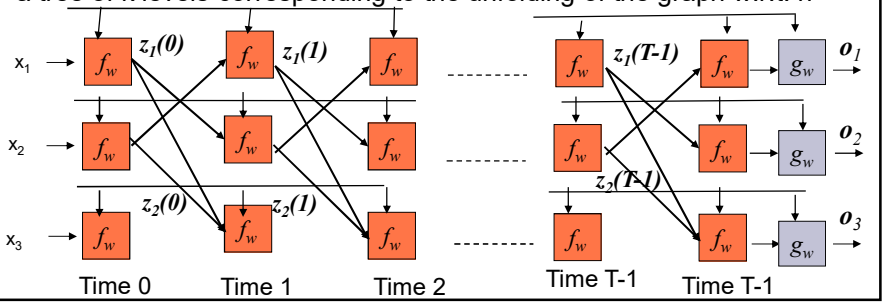
Intuitive idea of the proof

- We adopt the same reasoning we used for recurrent networks
  - If the aggregation function f_w can store into the state z the graph,
  - then the output function g_w can decode the state and produce any desired ouput from z
- But with graphs, there is a difference that it is worth to discuss

258

Unfolding tree

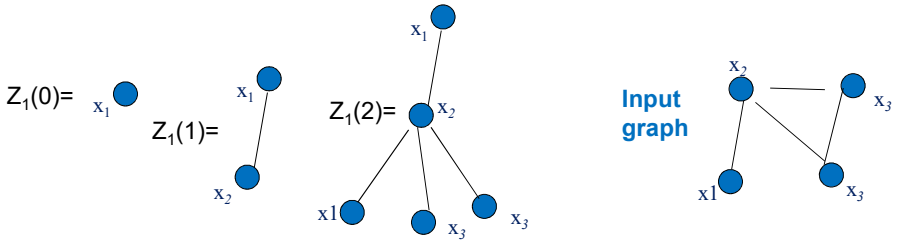
- The state $z_n(0)$ can contain only info about the features of node n
- The state $z_n(1)$ can contain only info about node n and of its neighboors
- The state $z_n(2)$ can contain only info about node n, about its neigboors and its neighboors of neighboors
- ....
- The best you can do, it is to store store in $z_n(k)$ the unfolding tree:
  a tree of k levels corresponding to the unfolding of the graph w.r.t. n



259

Unfolding tree

- The state $z_n(0)$ can contain only info about the features of node n
- The state $z_n(1)$ can contain only info about node n and of its neighboors
- The state $z_n(2)$ can contain only info about node n, about its neigboors and its neighboors of neighboors
- ....
- The best you can do, it is to store store in $z_n(k)$ the unfolding tree:
  a tree of k levels corresponding to the unfolding of the graph w.r.t. n



260

The consequences of the locality of computation

Two copies of the same node or two different nodes with the same features?

- The unfolding trees may contain copies of the same nodes due to cycles and the bidirectional links
- The aggregation function can aggregate the information from neighboors, but, since it is a local activity, the aggregation may not able to distinguish two copies of the same node from two different nodes with the same features

261

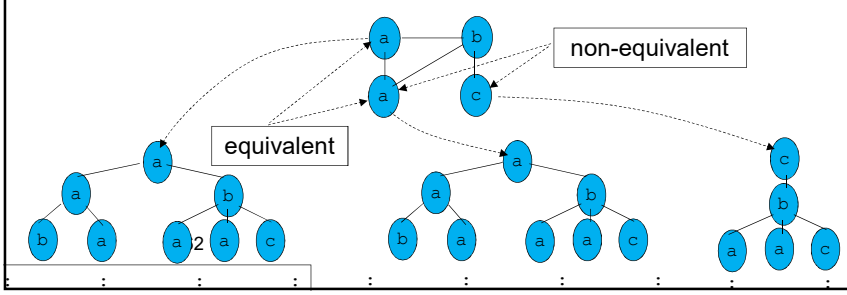
The unfolding equivalence

Unfolding tree

n The unfolding tree T_v^d is the tree obtained by unfolding the graph starting in v up to d levels

Unfolding equivalence

n two nodes are unfolding equivalent n~v if $T_n^d = T_v^d$ holds for every d



262

Functions that preserve the unfolding equivalence

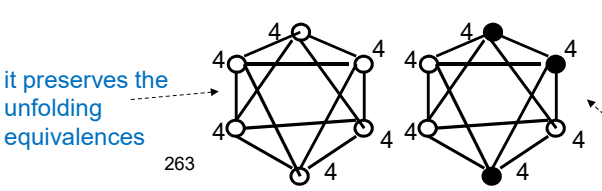
Functions that preserve the unfolding equivalence

- A function $\tau : \mathcal{G} \times \mathcal{N} \rightarrow R^m$ preserves the unfolding equivalence if

  n~v implies $\tau(G,n)=\tau(G,n)$

Example

- a function that preserves the unfolding equivalence has the some output on a uniform graph where all the node has the same labels



it preserves the unfolding equivalences

263

Black nodes = output 1
White nodes = output 0

it does not preserve unfolding equivalences

263

Thus, what GNNs cannot approximate

GNNs and the unfolding equivalence

- GNNs can approximate only the functions that preserve the unfolding equivalence: this result applies also to modern GNNs!
- such a behaviour may not be a limitation
  - two concepts having the same labels and the same relationships with the other concepts provide the same information:
  - why those concepts should be distinguished
- with an appropriate modification of the graphs, a functions may preserve the unfolding equivalence
  - f.i. when all the labels of the input graph are distinct labels

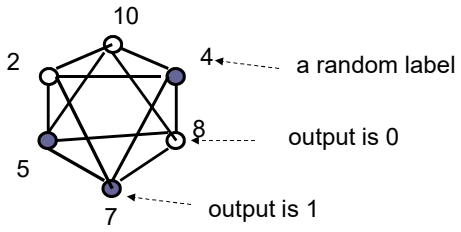
264

264

A funny experiment about GNN approximation properties

A GNN produces the same result on a uniform graphs but …

- if a noise is introduced into the labels, the nodes become distinguishable and, in theory, GNNs can implement any function
- is it possible to learn a GNN that return -1 for a half of the nodes (white nodes) and 1 for the other half (black nodes)?



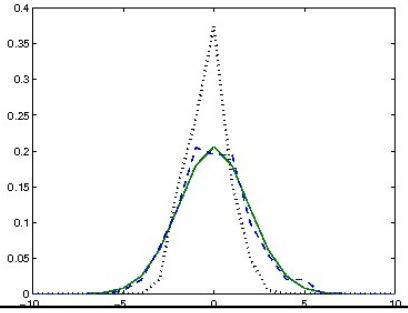
265

265

A funny experiment on GNN approximation properties

300 random uniform graphs

- random labels
- the error was measured by the difference between the desired number of black nodes on those obtained by the GNN
- results were compared with those obtained by random process and a feedforward neural network



266

266

Which aggregation function?

Advanced question

- We would like to get some details about the aggregation and the output function we should use

Main idea by Xu et al

- exploit Weisfeiler-Lehman test of isomorphism
  - An algorithm that produces iteratively a coding of the graph an allows to recognized whether two graphs are isomorph
- GNNs are sufficiently powerfull when they can implement Weisfeiler-Lehman test of isomorphism

267

Which aggregation function?

Xu et al.

- Let us assume that f_w is decomposed as

$$z_v(t) = \pi\big(z_v(t-1), \tau(\{z_{n(t-1)}: v \to n\})\big)$$

- and that there is a function θ that combine all the states of a graph to produce an output for the graph

$$\varphi(G)=\theta(z_{v1}(T),\ldots,)$$

(by this schema, we capture most of modern GNNs

268

Which aggregation function?

Xu et al.

- Let us assume that f_w is decomposed as

$$z_v(t) = \pi\big(z_v(t-1), \tau(\{z_{n(t-1)}: v \to n\})\big)$$

$$\varphi(G)=\theta(z_{v1}(T),\ldots,)$$

- If τ aggregates recursively the children features
- π, and θ are injective
- then the GNN produces an embedding of the graphs on which we can apply the Weisfeiler-Lehman test to recognize whether the graphs are isomorphic or not

269

Which aggregation function?

$$z_v(t) = \pi\big(z_v(t-1), \tau(\{z_{n(t-1)}: v \to n\})\big)$$

- If τ aggregates recursively the children features
- then, the aggregation function should have at least two layers (one hidden layer)
  - but most modern GNNs exploit just a single layer, e.g. in CGN

$$z_v(k) = \sum_{n \to v} relu\left(\frac{z_v(k-1)\,Wk}{|ne[v]||ne[n]|}\right)$$

  - The advantage is in the fact that using a single layer makes the architecture simpler, faster, and improves generalization
  - A big advantage for applications with a lot of features

270

Which aggregation function?

$$z_v(t) = \pi\big(z_v(t-1), \tau(\{z_n(t-1): v \to n\})\big)$$

$\varphi(G)=\theta(z_{v1}(T),\ldots,)$

- π, and θ are injective
  - But some modern GNNs exploit AVERAGE, MAX. In this case, π,/θ are not injective. e.g. in CGN

$$z_v(k) = \sum_{n \to v} relu\left(\frac{z_v(k-1)\,Wk}{|ne[v]||ne[n]|}\right)$$

271

Which aggregation function?

An example

- the average (normalization) may be useful to avoid that the gradient become smaller during back propagation
- but with average, you cannot count, e.g. the number of children of a nodes
  - If your target application is to recognize the users that have a lot of friends in a a social network, using AVERAGE in your aggragation function is nota good idea!

272

Generalization in GNNs

- From a theoretical view point, there are few results generalization in GNNs
  - One for the old GNN model (Scarselli, Tsoi, et al)
    - Based on Vapnik-Chernovenkis
    - The bounds are similar to the of other networks
    - Surprisingly, the length of unfolding in time does not play any role: this may be due to converge of the GNN model
  - One for modern GNNs (Garg et al.)
    - Based on Rademacher complexity (not studied here)
    - The bound depend on the lenth of unfolding
    - Similar to that or recurrent networks

273

Bound on VCD dimension of the orginal GNNs

p=number of paramters
N=number of graph nodes

Activation	Bound
Graphs (GNNs and non–positional RecNNs)	
Polynomial	O(p log(N))
Tanh,logsig, atan	$O(p^4N^2)$
Sequences (recurrent networks)	
Polynomial	O(p log(N))
Tanh,logsig, atan	$O(p^4N^2)$
Vectors (multialyer neural networks)	
Polynomial	O(p log(p))
Tanh, logsig, atan	$O(p^4)$

274

274

Bound on Rademacher complexity of some modern GNNs

d=brancing factor
r=state dimension
L=depth
m=sample size
γ= margin used in the margine loss

C = a sort of Lipschitz constant for aggregation

	GNNs	RNNs
C<1/d	$O\left(\frac{rd}{\sqrt{m}\gamma}\right)$	$O\left(\frac{r}{\sqrt{m}\gamma}\right)$
C=1/d	$O\left(\frac{rdL}{\sqrt{m}\gamma}\right)$	$O\left(\frac{rL}{\sqrt{m}\gamma}\right)$
C>1/d	$O\left(\frac{rd\sqrt{rL}}{\sqrt{m}\gamma}\right)$	$O\left(\frac{r\sqrt{rL}}{\sqrt{m}\gamma}\right)$

275

275

Generalization in GNNs in practice

- Depending on the application, the generalization in GNNs is difficult to control
  - Test distribution must be same as training distribution both for node features and node connectivity
  - For complex applications, a lot of data may be required

276

UNIVERSITÀ DI SIENA 1240

Bibliography

- Auer, P., Herbster, M., & Warmuth, M. K. (1996). Exponentially many local minima for single neurons. In *Advances in neural information processing systems* (pp. 316-322).
- Baldi, P., & Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, *2*(1), 53-58.
- Bartlett, P. L. (1998). The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE transactions on Information Theory*, *44*(2), 525-536.

277

UNIVERSITÀ DI SIENA 1240

Bibliography

- Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157-166.
- Bianchini, M., Gori, M., & Maggini, M. (1994). On the problem of local minima in recurrent neural networks. *IEEE Transactions on Neural Networks*, *5*(2), 167-177.
- Bianchini, M., & Gori, M. (1996). Optimal learning in artificial neural networks: A review of theoretical results. *Neurocomputing*, *13*(2-4), 313-346.
- Bianchini, M., & Scarselli, F. (2014). On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems*, *25*(8), 1553-1565.
- Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, *39*(3), 930-945.

278

UNIVERSITÀ DI SIENA 1240

Bibliography

- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* *2*(4), 303-314.
- Hammer, B. (2000). On the approximation capability of recurren neural networks. *Neurocomputing*, *31*(1-4), 107-123.
- Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, *2*(5), 359-366.
- Funahashi, K. I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural networks*, *2*(3) 183-192.
- Garg, Vikas K., Stefanie Jegelka, and Tommi Jaakkola. "Generalization and representational limits of graph neural networks.", ICML, 2020.

279

UNIVERSITÀ DI SIENA 1240

Bibliography

- Gori, Marco, and Alberto Tesi. "On the problem of local minima in backpropagation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.1 (1992): 76-86.
- Gori, M., & Scarselli, F. (1998). Are multilayer perceptrons adequate for pattern recognition and verification?. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(11), 1121-1132.
- Huang, G. B., Zhu, Q. Y., & Siew, C. K. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, *70*(1-3), 489-501.
- Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." ICML, (2016).
- Judd, J. S. (1990). *Neural network design and the complexity of learning*. MIT press.

280

Bibliography

- Lawrence, S., Tsoi, A. C., & Giles, C. L. (1996, June). Local minima and generalization. In *Neural Networks, 1996., IEEE International Conference on* (Vol. 1, pp. 371-376). IEEE.
- Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.
- F Scarselli, AC Tsoi, M Hagenbuchner, The Vapnik–Chervonenkis dimension of graph and recursive neural networks, Neural Networks 108, 248-259
- F Scarselli, M Gori, AC Tsoi, M Hagenbuchner, G Monfardini, Computational capabilities of graph neural networks,IEEE Transactions on Neural Networks 20 (1), 81-102, 90, 2008
- F Scarselli, M Gori, AC Tsoi, M Hagenbuchner, G Monfardini, The graph neural network model, IEEE Transactions on Neural Networks 20 (1), 61-80

281

Bibliography

- Shawe-Taylor, J. (1993). Symmetries and discriminability in feedforward network architectures. *IEEE Transactions on Neural Networks*, *4*(5), 816-826.
- Yu, X. H., & Chen, G. A. (1995). On the local minima free condition of backpropagation learning. *IEEE Transactions on Neural Networks*, *6*(5), 1300-1303.
- Vapnik, V. N. (1999). An overview of statistical learning theory. *IEEE transactions on neural networks*, *10*(5), 988-999.
- Xu, Keyulu, et al. "How powerful are graph neural networks?." ICLR,2019
- Zhang, Chiyuan, et al. "Understanding deep learning requires rethinking generalization." *arXiv preprint arXiv:1611.03530* (2016).

282

UNIVERSITÀ DI SIENA 1240

Thank you for your attention!

283

UNIVERSITÀ DI SIENA 1240

Lecture of July 21th

- The lecture will start at 10.00 am
- We will have a test
  - Written
  - Close answers
  - About 1hour time to complete
- The test is mandatory for students of Smart Computing PhD who want to have to pass an exam to have the credits,
  - but it is open to anybody
- After the test, we will discuss the responses, thus this will be used as a course summarization

284