Deep neural networks (DNNs)

178

Deep neural networks (DNNs)

Definition ver 1.0

- Just networks with many layers

Up to beginning of 2000, 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!!

179

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

180

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, …

181

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

182

Convolutional neural networks: convolutional layer

Kernel filters in image processing

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

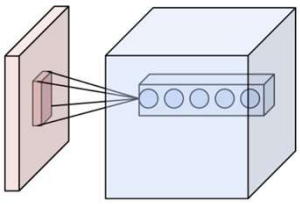
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$ Edge Detection $$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$ Blur

- 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!!

183

Convolutional neural networks: convolutional layer

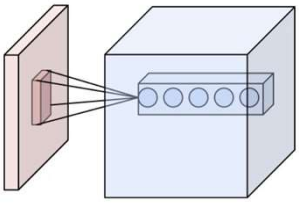


Convolutional layers

- 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

184

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

185

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

186

Convolutional neural networks: fully connected layer

Fully connected layers

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

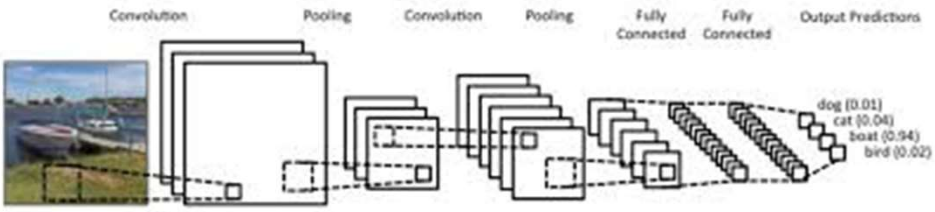
- Intuitively, fully connected layers allow to combine and reason about the extracted features in order to take the final decision

187

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



188

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?

189

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 also applying a number of tricks
    - Pooling
    - Weight decay
    - Use validation to stop
    - Dataset aumentantion
      - Images can be rotated, scaled, ….

190

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

191

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

192

The underlining idea

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

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

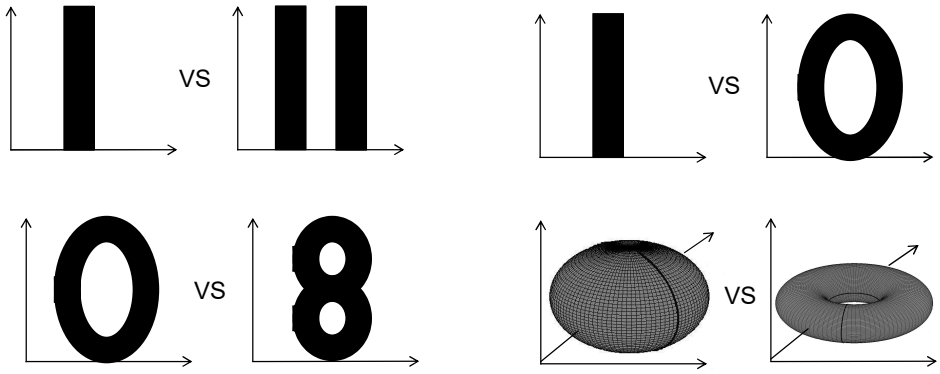
The idea

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

193

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?



194

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

195

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

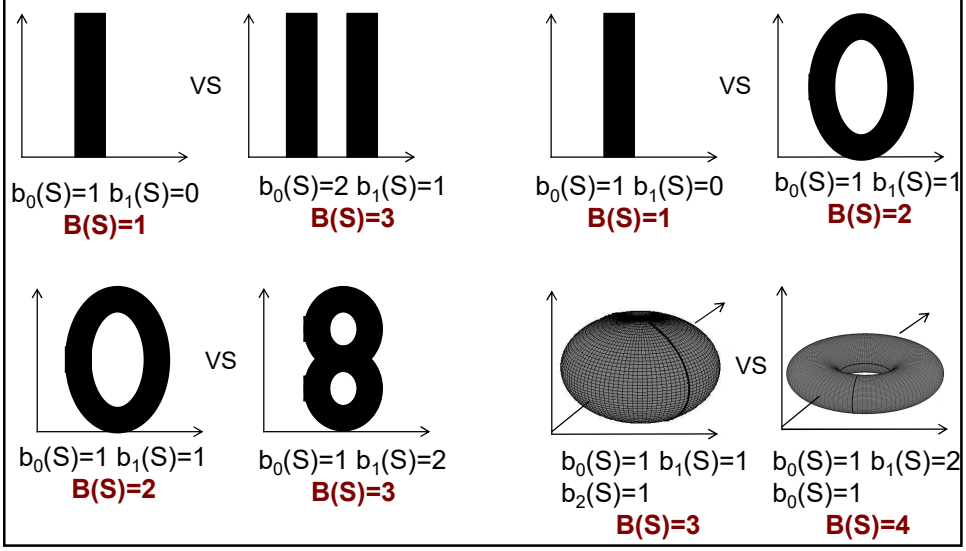
- We will measure the complexity of the function f_N implemented by a neural network N by

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

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

196

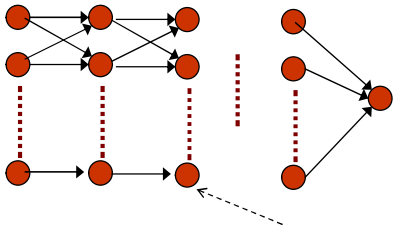
Betti numbers: examples



197

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

198

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	

199

UNIVERSITÀ DI SIENA 1240

Analysis of the results

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

- always grows exponentially

200

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	

201

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

202

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	$\boldsymbol{h}$
1	polynomial	$(2+r)(1+r)^{n-1}$	n	r
1	arctan	$(n+h)^{n+2}$	n	$\boldsymbol{h}$
many	arctan	$2^{h(2h-1)} \bullet (nl+n)^{n+2h}$	$n,\boldsymbol{h},l$	
many	tanh	$2^{h(h-1)/2} \bullet (nl+n)^{n+h}$	$n,\boldsymbol{h},l$	
many	polynomial	$(2+r^l)(1+r\,l)^{n-1}$	$n, l, \boldsymbol{h}$	r
		Lower bounds		
1	sigmoid	$((h-1)/n)^n$	n	$\boldsymbol{h}$
many	sigmoid	2^{l-1}	$l,\boldsymbol{h}$	
many	polynomial	2^{l-1}	$l,\boldsymbol{h}$	

203

UNIVERSITÀ DI SIENA 1240

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*

204

UNIVERSITÀ DI SIENA 1240

An intuitive explanation of the advantage of deep architectures

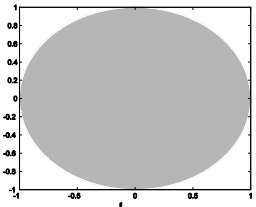
Notice that

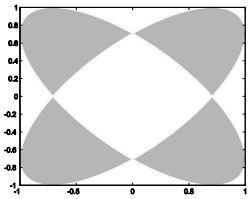
- A layered network implements a function
  $f_N = g_1 \circ g_2 \circ .. \circ g_l \circ t$
  - g_k is the function implemented by layer k
  - $\circ$ is the function composition operator
- If $f_N = g_1 \circ t$, then f_N behaves as t on all the regions $A_{1,..}, A_s$ where $g_1(A_k) = R^n$
- With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

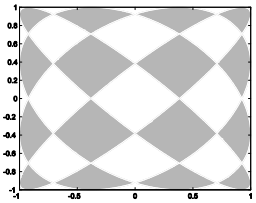
Deep networks can replicate more easily the same behavior on different regions of the input domain

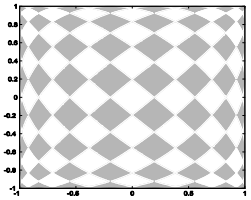
205

An example of the advantages of deep networks


t


g◦t


g◦g◦t


g◦g◦g◦t

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

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

206

Learning in DNNs with a lot of parameters

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"? (weight decay, …)
  - The networks easily reaches 100% on train set,
  - but of course, the generalization may be lower (surprising, it still generalize)
- 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

207

Over-parametrized networks

It has been proved that Zou et Alt.

- Deep networks with Gaussian random initialization on L layers
- n patterns are separated at elast by α
- when the number of hidden nodes per layer is at least $\Omega(n^{14}L^{16}/\alpha^4)$
- Then gradient descent can achieve zero training error within $O(n^5L^3/\alpha)$ iterations,

- No local minima, a defined number of iterations

208

Over-parametrized networks

Intuitive explanations (attempt)

- Deep networks contains modules which solve sub-problems: those modules may not be over-parametrized
- Gradient descent produce an implicit smooting, since it uses patterns to iteratively defines a solution: in this way, the weight solution is close to the starting point and it small as suggested by patterns
- Convolutional networks have a limited approximation capability, due to the presence of convolutions that capture images simmetries

- Researchers are currently try to better understand the actual role played by over-parametrization

209

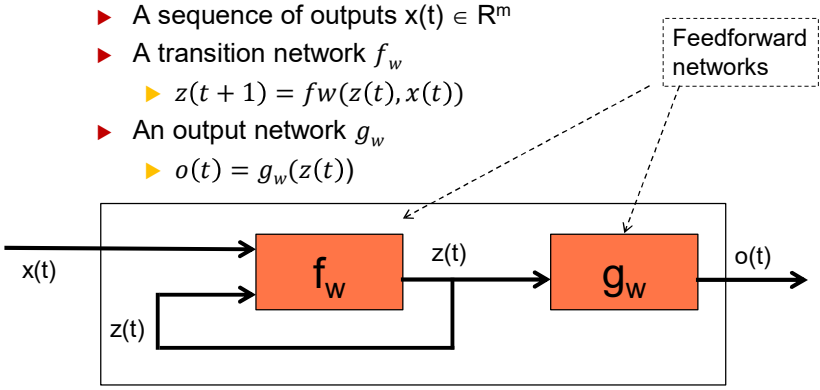
Recurrent neural networks

210

Recurrent neural networks (RNNs)

Dynamical neural networks

- An internal state $z(t) \in R^s$
- A sequence of inputs $x(t) \in R^n$
- A sequence of outputs $x(t) \in R^m$
- A transition network f_w
  - $z(t+1) = fw(z(t), x(t))$
- An output network g_w
  - $o(t) = g_w(z(t))$



211

Training recurrent neural networks

The train set

- A pattern is a pair of sequences of inputs and desired outputs
- $L = \{(\bar{x}, \bar{t}) | \bar{x} = x(0), x(1), \ldots x(T), \bar{t} = t(1), .. t(T)\}$

212

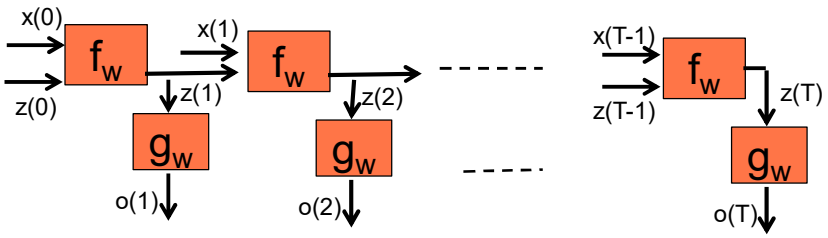
Training recurrent neural networks

Unfolding network

- A copy of f_w, g_w for each time instance, connected in sequence

The unfolding network is a feedforward network (with shared weights)

- The unfolding network can be trained by standard backpropagation (just remember to accumulate gradients)



213

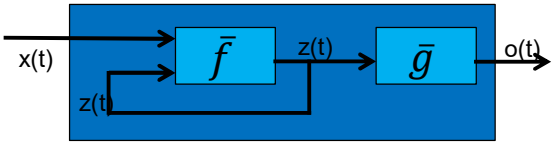
Approximation capability of RNNs

RNN approximation ver 1.0

- given dynamical system, can it be approximately simulated by a RNN?

Formally, is there a RNN that simulates the following?

- a transition function $\bar{f}$
- an output function $\bar{g}$

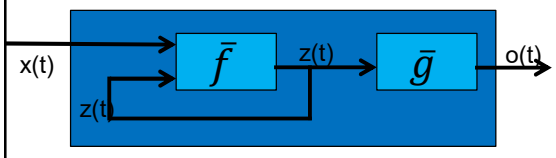


214

Approximation capability of RNNs

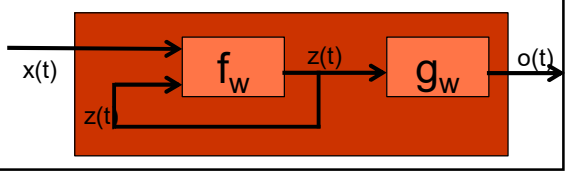
Obvious answer

- Yes
  Just take neural networks that approximate $\bar{f}$ and $\bar{g}$



Target dynamic system

RNN



215

Simulating a dynamical system

Obvious answer

- Yes
  Just take neural networks that approximate $\bar{f}$ and $\bar{g}$

But notice

- We implicitly assumed the state dimension is known and you can design your network with the same internal state dimension
- The transition and output networks must have at least one hidden layer
- The sequences of outputs generated by the two dynamical system may diverges with time

216

Approximation capability

An advanced question

- Let us consider a sequences of patterns $\overline{x(T)} = x(0), x(1), \ldots x(T)$ and focus the attention on a function t that returns an output $t(\overline{x(T)}) = o(T)$
- Can a RNN approximate t?
- Notice: no assumption is made on the existence of a dynamical system able to implement t …. we are studying that

An intuitive answer

- Yes, provided that
  1. The transition network can code the input sequences (or all the relevant information) and store into the state z(t)
  2. The output network can decode such a representation and produce the output
- If the coding exists, the thesis is true by universal approximation

217

Approximating sequences

- Yes, provided that
  1. The transition network can code the input sequences (or all the relevant information) and store into the state z(t)

Simple case: the dimension of the state must be large enough

- It must hold s > T*n
- just copy inputs to the state

The general case, s<T*n

- a set of integers $v_1, .., v_k$ can be coded with the real number $0.v_1, ..v_k$
  - E.g. 0,1234 is a coding of 1, 2, 3, 4
- a set of real number $v_1, .., v_k$ can be approximately coded with the real number $0.[v_1] ..[v_k]$, where [.] is the rounding
  - E.g. 0,1234 is a coding of 1.12, 2.15, 3.41, 4.32

Approximating sequences

It has been proved that (Hammer)

- Let us consider a function t that takes in input sequences of patterns $\overline{x(T)} = x(0), x(1), \dots x(T)$ and returns an output $t(\overline{x(T)}) = o(T)$
- t can be approximated by a RNN which is feeded with x(i) and return o(i) for each i
  - If t is measurable and $x(0), x(1), \dots x(T)$, are reals, then the approximation can be obtained in probability
  - If t is continuous and $x(0), x(1), \dots x(T)$, are integers, then the approximation is w.r.t. sup norm

RNN approximation in practice

- when s<<T*n, the coding function exists, but
  - it is complex
  - It is very sensible to noise
  - .. so it is difficult to learn
- In real life tasks
  - only a small part of the information in inputs is useful
  - Inputs are recursively processed
  - Inputs belong to a sub.mainfold
  - … information to be stored is much smaller and a much smaller state dimension is required

220

Learning capability of RNNs

Intuitively

- Known results recall that of feedforward networks

It has been proved that (Bianchini et alt)

- a RNN with one layer network
- error has no local minima, if
  - the matrix of the inputs $x^1(0)$, …, $x^1(T)$, $x^2(1)$… $x^2(T)$, … is full-rank
  - Thus, the total number of time instances cannot larger than input dimension

221

Learning capability of RNNs

Intuitively

- Known results recall that feedforward networks

It has been proved that (Bianchini et alt)

- a RNN with one layer network
- error has no local minima, if
  - The sequences are linearly separable
  - The weights of links connecting states to states are positive

222

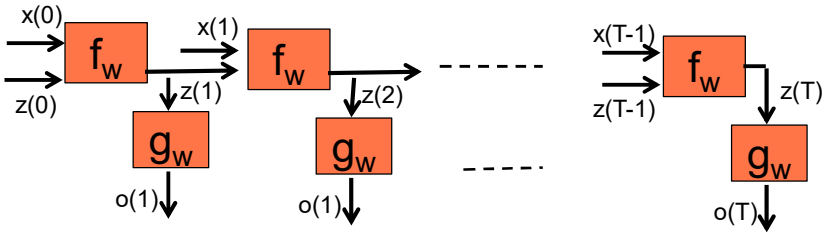
Long-term dependencies

Intuitively

- It is difficult to learn a RNN when the at time T depends on the input at time t and t<<T

Common explanation

- The gradient become smaller and smaller when propagated through the layers of the unfolding network



223

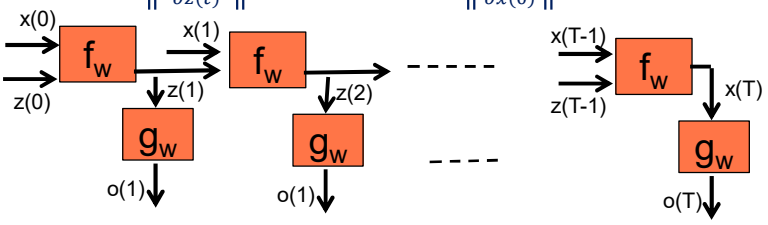
Common explanation

Remember

- $e_w(T) = (l - ow(T))^2$
- $\frac{\partial ew(T)}{\partial x(0)} = \frac{\partial ew(T)}{\partial z(T)} \frac{\partial z(T)}{\partial Z(T-1)} \frac{\partial z(T-1)}{\partial z(T-2)} \cdots \cdots \frac{\partial z(1)}{\partial x(0)}$

The term $\frac{\partial z(t+1)}{\partial z(t)}$

- A sxs matrix measuring how the input state of f_w affects its output
- When $\left\|\frac{\partial z(t+1)}{\partial z(t)}\right\|$<1 and T is large $\left\|\frac{\partial e_w(T)}{\partial x(0)}\right\|$ is close to 0

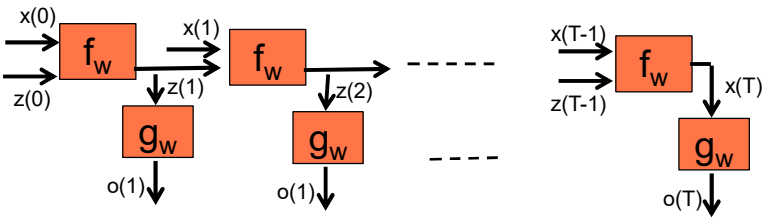


224

Common explanation

$\left\|\frac{\partial z(t+1)}{\partial Z(t)}\right\|$ is small where the function f_w is flat

- small weights
- saturated neurons

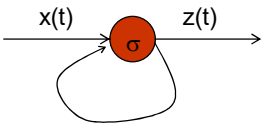


225

Long-term dependencies: some experiments

Bengio, Frasconi, et alt.

- The problem: learning to latch
  - An input sequence x(0)…x(T)
  - Two classes to be reconignized which depend only on x(0)
  - The expected output at time T o(T)>0 for class A, o(t)<= fo class B
- The network
  - A RNN with a single hidden neuron
  - σ=tanh
  - $z(t)=w\,\sigma(z(t-1))+x(t)$

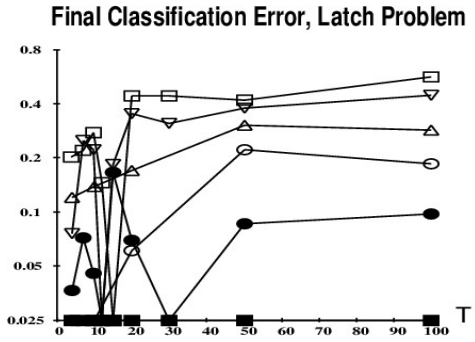


226

Long-term dependencies: some experiments

Bengio, Frasconi, et alt.

- The classification task is learned only for small T (T<20)
- Some tricks with learning algorithms improved only a few the results

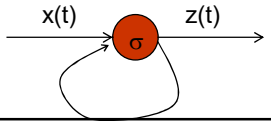


227

Long-term dependencies: the theory

Bengio, Frasconi, et alt.

- .If w>1 then the system has two attracting stable states, a positive z^+, and a negative stable state z^-
  - Intuitively: the system is able to store some information only if w>1
  - This is true in general, if the weights are too small then the RNN has only one stable point!!|!
- In this case, If z(0)>0, z(0) is large enough and |x(t)| not too large then z(T)>0 (the converse hold if z(0)<0)
  - Intuitively: the system resets the stored information when the input is large or small enough
  - The information storing is robust for small inputs

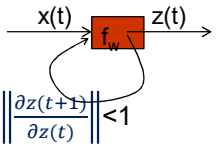


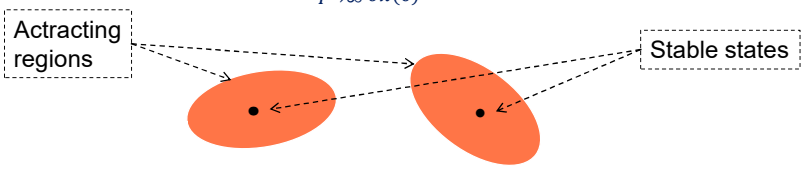
228

Long-term dependencies: the theory

Bengio, Frasconi, et alt.

- A more general network with any number of neurons
  - $z(t)=f_w(z(t-1))+x(t)$
- The system may have several attracting points
- There are regions close to those points where $\left\|\frac{\partial z(t+1)}{\partial z(t)}\right\|<1$
- Those regions
  - make information latching robust
  - But if inputs do not move the state from an attracting regions, then $\lim_{T\to\infty}\frac{\partial e(T)}{\partial x(0)}=0$





229

Long-term dependencies: other explanations

Loading problem is difficult with many inputs

- The learning algorithm is a search algorithm in network space: with many inputs, the search space is large
  - which are the inputs affecting the output?

Antagonist goals

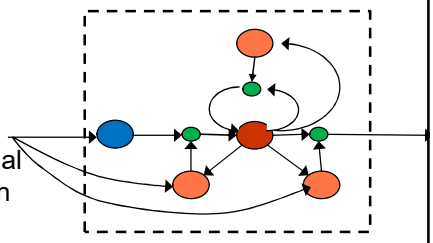
- weights must be large to have several attracting regions:
  - but large weights decrease generalization capability and/or saturate sigmoids and/or make gradient oscillate
- There must exist regions where $\left\|\frac{\partial z(t+1)}{\partial z(t)}\right\|$<1 holds to store information in a robust way
  - But $\left\|\frac{\partial z(t+1)}{\partial z(t)}\right\|$<1 makes make gradient small
  - Small regions are difficult to reach

230

Long short-term memory: a solution

The idea

- keep $\left\|\frac{\partial z(t+1)}{\partial z(t)}\right\|$ equal to 1
  - this makes the error signal to remain close 1 through several time steps



LSTM cell

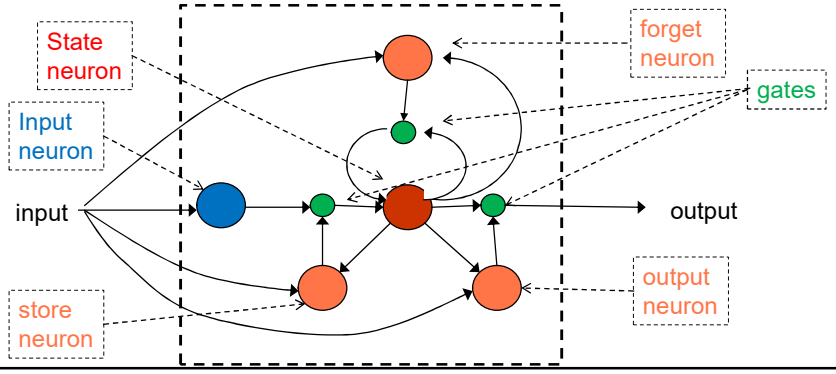
- a neuron to store the state
- an input neurons
- a store neuron and a gate to decide whether to store input
- a forget neuron and a gate to decide whether to reset the state
- An output neuron and gate to decide whether to output the state

231

Long short-term memory

LSTM cell

- neurons use sum and sigmoidal activation
- gates use just product



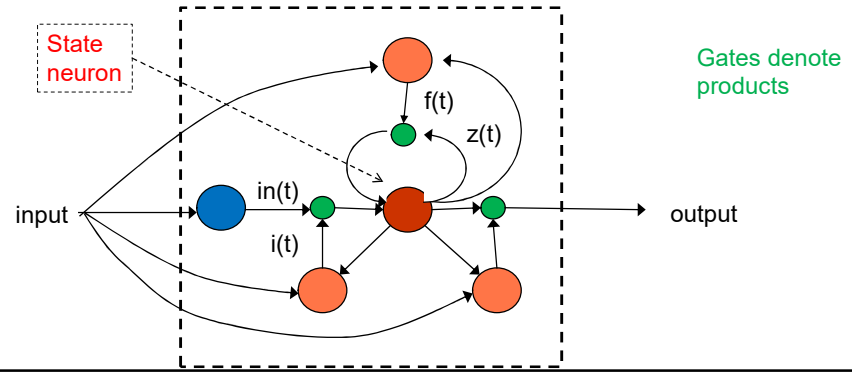
232

Long short-term memory

The state

- f(t) is 0 to forget, it is 1 to remember
- i(t) is 1 to store input, it is 0 to skip

$z(t+1) = f(t)z(t)$+i(t) in(t)



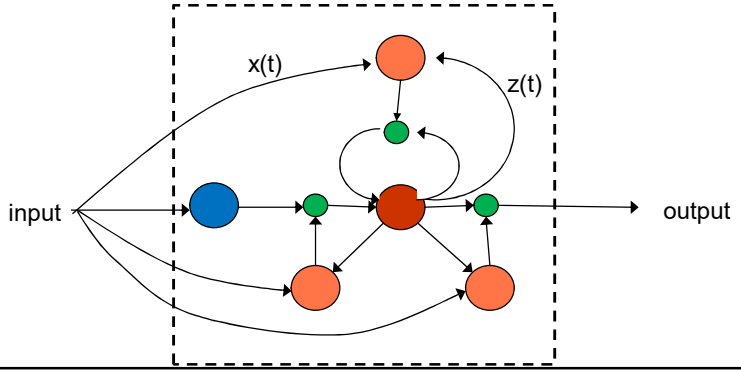
233

Long short-term memory

Store, forget and output neurons

- They are standard neurons with sigmoidal activation function and weights, e.g.

$$f(t) = \sigma(w_z z(t) + wxx(t))$$

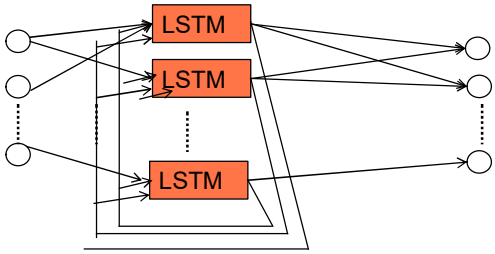


234

Long short-term memory

LSTM networks

- constructed connecting LSTM cell
- They can be mixed with standard neurons
- common architecture: a standard input layer, a hidden layer of LSTM cells, a standard output layer



235

Long short-term memory

LSTM

- can store input for hundreds/thousands of instances
- Successfully applied in several application domains, e.g. language translation

236

Long short-term memory: why do they work?

Explanation ver 1.0

- When the forget gate is 1, then we have $\left\|\frac{\partial z(t+1)}{\partial z(t)}\right\|$=1

$z(t+1) = f(t)z(t)$+i(t) in(t)

Another Explanation

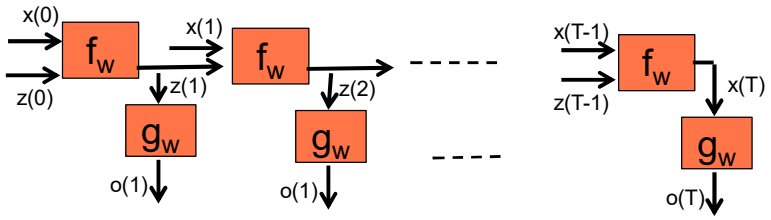
- LSTM assumes that
  - the problem can be solved by storing/forgetting inputs/states
  - The decision whether to store/forget is simple (linearly separable, implementable by a single layer net)
- RNN does not make any assumption
- LSTM are better than RNN if the assumption is satisfied
  - The search space is reduced!

237

Generalization capability of RNNs

Intuitively

- VCD of RNNs can be studied by observing the unfolding network
- However, such an assumption does not consider the implications of the weight sharing
- Current results may suffer of this limitation



238

Generalization capability of RNNs

Neural networks with *p* parameters, T time steps, bounds for the order of growth of VCD(f_w)

- RNNs with piecewise polynomial activation function
  Upper bound: $O(Tp^2)$,
  Lower bound $o(p^2)$
- RNNs with, tanh, logsig, atan activation function
  Upper bound: $O(T^2p^4)$
  Lower bound $o(p^4)$

239