11-785 08/07/2020

LIS-RNNs pt 3

- watch S.2019 video

Review

- long-term dependence → recurrent structures
- short-term dependence → iterated structures

BR: introducing self-reference (!) - are there deeper implications here.

- investigate.
- RNNs trained by minimising divergence between sequence of outputs and desired outputs. (not between instantaneous inputs and outputs)

(*) Primary topic:-
- How to backprop with divergences over sequences

(*) Story so far-stability

(A1) - clarify these metaphors.

- RNNs can be unstable
- Have to deal with issues of 'blowing up', but also 'saturation'
- tanh() more reasonable.

(*) vanishing gradients

- Amissive for deep networks in general
- Backprop → gradients vanish/explode.
- e.g. 1st layer MLP
  - see magnitude of gradient wrt params of network (near beginning of network)
- (L2) Propagation of info to subsequent layers.
  - less than required no. of neurons limits to 'capture' pattern complexity; layers downstream cannot recover this 'missed info'
  - for deeper layers to be able to recover patterns; want info to be 'pushed' downstream (i.e. by not using thresholds) and activations which 'carry info' about distance from boundary.
- (*) We see this again → more informative activations might be ReLU or soft ReLU.
- (*) other activations e.g. tanh, sigmoid have saturation behaviour, beyond specific distance from boundary, 'info about where you are' is 'lost'. (A2) too informal.

(*) Even with ELU activations, gradients can vanish after only a few layers of backprop

(?): How is this remedied?

(*) Long term dependency problem

- Network 'forgets' in a manner that depends primarily on weights rather than inputs.
- We really want a design where system 'retains memory' until it no longer has to.
- Sentence dependency example - want behaviour that is triggered not only on what is 'remembered', but also on what next expected input or output must be.
- i.e. input-triggered response rather than purely time-triggered response

↳ Cue LSTM

(*) Long Short Term Memory

- Still mathematically a complex activation that suits our design criteria
- ABR: Addresses problem of input dependent memory behaviour.

(*) LSTM-based architectures

- Identical to RNN arch.
- Each of 'boxes' is a layer (many units) - remember not. conventions!

(*) Bi-directional LSTM

- If you agree LSTM is a variant on RNNs, can have things like bidirectional LSTM.

(*) Key issue

1) How do we define divergence?

2) How do we compute outputs?

(*) today - focus on unidirect. structures ; can be generalised easily to bidirectional.

- focus on :-

1) Network architectures

2) Synchrony :-

i) Time synchronous -

ii) Order synchronous

(*) variants on RNNs. (A3) - see slides ; acquaint with architecture

- one-to-one :
- many-to-many :
- many-to-one :
- order synchronous : - get a sequence of outputs in response to sequence of inputs
  - order correspondence between input and output.
  - 1-to-1 correspondence between input and output not present.

(time asynchronous).

- many-to-many : generate output sequence after processing input
- one-to-many : e.g image captioning.

(*) variants on RNNs / eg MLP

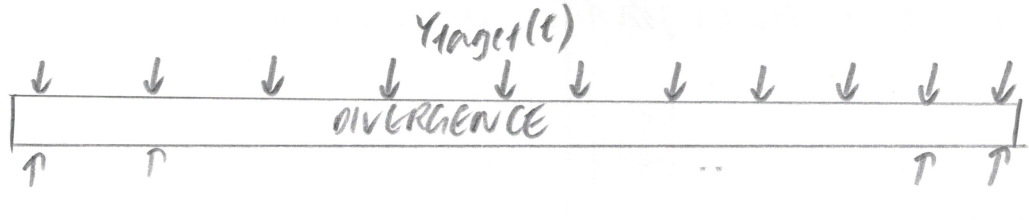
1-to-1

- no recurrence in model
- one input → one output ; unique
- why is this included in RNNs ?
- we can still view divergence over sequence of outputs rather than as individual divergences over outputs.

- OR : Point he is trying to make is to rationalise the view of divergence over a sequence of outputs ; and to consider the decomposition of divergence over a sequence into that of over individual outputs as a special case of a more general phenomenon.

- weighted sums.

(*)



- gradient backprop at each time $\nabla_{Y(t)} Div(Y_{target}(1,\dots,T), Y(1,\dots,T))$ (*) - more general case.

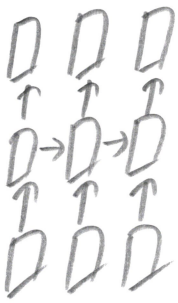
(previous)
- Assumption: $Div(Y_{target}(1,\dots,T), Y(1,\dots,T)) = \sum_t w_t Div(Y_{target}(t), Y(t))$

(simplifying) as an MLP. $\nabla_{Y(t)} Div(Y_{target}(1,\dots,T), Y(1,\dots,T) = w_t \nabla_{Y(t)} Div(Y_{target}(t), Y(t))$

- w_t typically set to 1.0.
- (?) Surely $\sum_t w_t \nabla_{Y(t)} Div(Y_{target}(t), Y(t))$?

(*) variants on RNNs (time synchronous net)

many to many



- below notation/modelling language
- network - one output for each input. ; one-to-one corres.
- POS tagging
- in practice bidirectional → future informs about current time.
- assignment of one tag depends on others ('recurrent rel')

- unidirectional time synchrous - process input left to right, produce output
- Bidirect. ——— " ——— process left to right (forward net)
  process right to left (backward net)
  combine hidden outputs to prod. one out. per input symbol

↓ used in practice (makes more sense)

(*) How do we tramtle input?

- BPTT.

(*) Given a collection of training instances comprising input sequences and output sequences of equal length, with one-to-one corresp.

- $(\underline{X}_i, \underline{D}_i)$ where $\underline{X}_i = (X_{i,0}, X_{i,1}, \dots, X_{i,T})$
  $\underline{Y}_i = (O_{i,0}, O_{i,1}, \dots, O_{i,T})$

(*) input - sequence of vectors also
output - sequence of vectors also.

(*) Backprop through time

(*) (5th): divergence is computed between network sequence of outputs and desired —— " ——

(5th): Not sum of divergences at indiv terms; unless defined that way

(*) compute $\nabla_{Y(t)} DIV$ for all t (1st step)

(*) Definition of the divergence DIV is very general

(*) can become complex

(*) time-synchronous recurrence

(*) simplifying ass: - sequence divergence is simple additive sum of divergences over individual time instances

- Equations: see earlier : Global divergence as sum of local.

(*) Typical div. for class is cross-entropy

(*) Simple recurrent model - text modelling

- Generate model that can predict next character from a seq. of characters / words
or word

(*) use one-hot-encoding

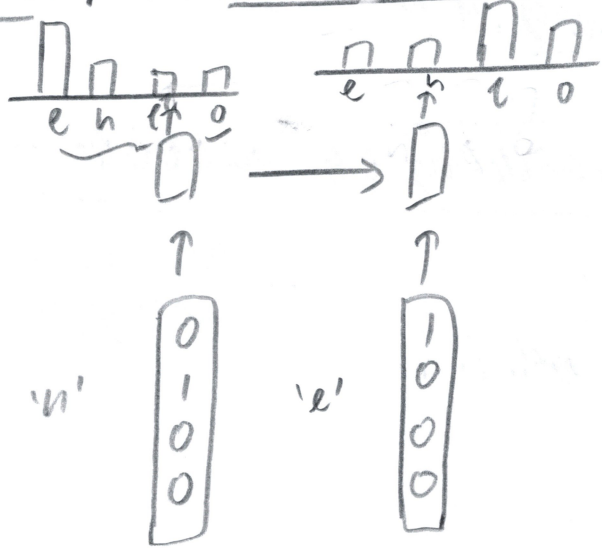
- softmax outputs class probabilities

input - one-hot vectors (embeddings?)

'hello' output - prob distn over characters

- peek at target char.

e.g.



$$\begin{pmatrix} e \\ h \\ l \\ o \end{pmatrix}$$

(*) Training

- Input: symbols as one-hot vectors - dimensionality is vocab size

- Output: probability distn over symbols

$$Y(t,i) = P(V_i | w_0, w_1, \ldots, w_{t-1}) \qquad V_i - i^{th} \text{ symbol in vocabulary}$$

- Divergence: $DIV(Y_{target}(1,\ldots,T), Y(1,\ldots,T)) = \sum_t CE(Y_{target}(t), Y(t))$

$$= -\sum_t \log Y(t, w_{t+1})$$

↳ prob. assigned to next correct word

(*)

y_1	y_2	y_3	y_4	network
0	0	1	0	target
e	h	l	0	
		T		

$-\sum_i d_i \log y_i$

$$CE = -(0 \log y_1 + 0 \log y_2 + \log y_3 + 0 \log y_4) = -\log y_3$$

(*) Cross-entropy is -ve log probability that was been assigned to target output

(*) Overall divergence is sum of this term over all time instances

(*) Brief detour - language models

(*) Language modelling using RNNs

OR: come up with modelling structures for prediction of lang.

one-hot encoding: Pre-specify a vocabulary of N words in a fixed lexical order.

- Distinction between representational granularity e.g. 100,000 English words vs 100 characters in English

(*) Predicting words

$w_n = f(w_1, \ldots w_{n-1}) \qquad w_1, \ldots, w_{n-1} \in \mathbb{R}^N$

(Nx1) - one hot vectors

(*) dimensionality of one-hot vector if using 100,000 English words

↳ $\mathbb{R}^{100,000}$

- inputs $w_1, \ldots, w_{n-1}$ are v high-dim, sparse

(*) unit cube 2^{100}, $2^{100,000}$ corners

(AA) - review this dimensionality a little

- need a more detail/agony

- occupying 100 dims, 100,000 words

OR: what vol. of space is entire collect. of inputs occupying? 0 (?)

(*) one-hot representation

- uses N corners of 2^N corners of unit cube
- density of points $O\left(\frac{N}{2^N}\right)$
- inefficient use of dimensions

(*) OR: why use one-hot if it uses dimensions inefficiently?

(AT) - semantic agnostic rep.
- not assigning rel. importance to words/going on symbols
- ensuring that distance between 2 words is same regardless of what words is.
- no imposition of closeness/distance or language → (AK) - develop with reading
- what distance?

- no assumption about relationship between words or relative importance of words; use of space is inefficient.

(*) Solution to dimensionality

- project to lower dimensional subspace: $O\left(\frac{N}{2^N}\right)$
- volume capacity not improving; but density is improving by many orders of mag.

(*) But reframed q: - what is projection/hyperplane such that positions/distances between points 'makes sense'.

(*) Learn/estimate the plane

- If done so 'properly', distances between projected points will capture semantic relations between words.

(*) Projected word vectors

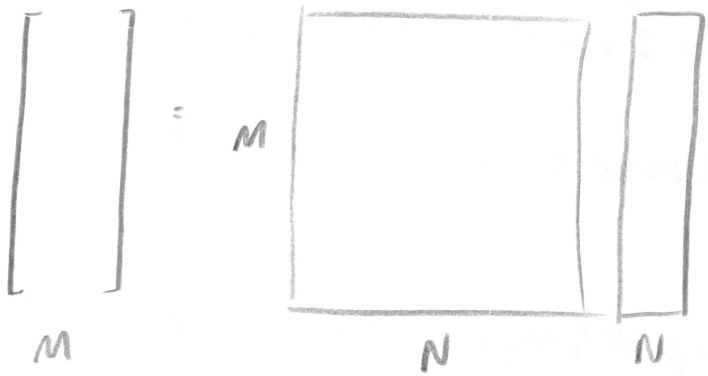
(*) Project one-hot word vectors $\underline{w}_i \in \mathbb{R}^N$ into a lower-dim. subspace; see slides → project comes

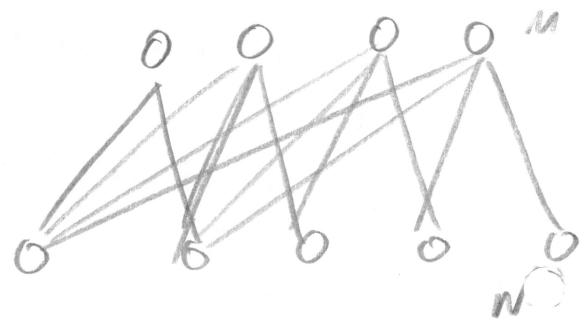
(*) Conduct following trans.

- Projection matrix: $P \in \mathbb{R}^{M \times N}$
- one hot word vector: $\underline{w}_i \in \mathbb{R}^N$
- Projected word vector: $P\underline{w}_i \in \mathbb{R}^M$ → use $\underline{w}_n = f(P\underline{w}_1, P\underline{w}_2, P\underline{w}_3, \ldots, P\underline{w}_{n-1})$

- estimate/learn P using an appropriate objective

(*) 'Projection'





(a): - This simple linear transformation P can be placed within an NN context.

- implemented as a layer of M neurons with linear activations
- Transforms that apply to individual words are all M-neuron linear activ. subnets with tied weights (as P is same for all word-vectors) (3)

(?): Is projection matrix P same $\forall$ word vectors $\underline{w}_i$?

(*) Predicting words - TDNN

- Predict each word based on past N words -see architecture
- Neural probabilistic language model (Begio et al. 2003)
- Hidden laye has tanh() activation, softmx

(*) Note the projection unit
- Estimate/learn low-dim representation PW of words

(*) Alternative models

(*) soft BOW - Predict word based on words in immediate context
- No consideration of specific position

(*) skip-grams. Predict adjacent words based on current word.

(*) Embeddings. example

- Mikolov et all (2013)
- Some Illustrates mapping from country → capital city capturing "is the capital of".
- cherry picked

(*) Generating language: The model

- use all perious words to generate next word

$W_n = f(PW_1, \ldots, PW_{n-1})$ $W_{10} = f(PW_1, \ldots, PW_9)$ etc.

(*) Generating language: synthesis

- will get a probability distribution over words in 4^{th} position
- Illustrates generation of text. ~ - prediction.
- select most likely word $\tilde{W}_4$
- use that as additional 'input'; then predict $\tilde{W}_5 = f(PW_1, \ldots P\tilde{W}_4)$ etc.
- practically, could go on indefinitely. But accuracy?

(?) Do feed raw probability vectors or post-process before feeding into next RNN step ?

(*) variants on RNN (sequence classification)

- e.g. speech recognition

  IN: sequence of feature vectors (mel spectra)

  OUT: phoneme ID at end of sequence

  - vdim out prob vector
  - N - no. of phonemes

(*) in this setting we have other outputs, just ignored, consider output at (produced every cycle) final time.

(*) inference: Forward Pass

- Review slides (A6) suppl.

- $DIV(Y_{target}, Y) = CE(Y(T), phoneme)$

- Divergence only defined at final input.

(*) Training (@)

(*) need to exploit unrolled inputs

- task details determine / guide definition of "divergence" (sequence)

$$DIV(Y_{target}, Y) = \sum_t w_t CE(Y(t), phoneme)$$

- speech/phoneme recog → equal weights w_t
- question answering → w_T high, further back weights low

(*) variants on RNNs (order synchronous, time asynchronous seq-to-seq)

- exact location of output unknown a priori