11-785 24/06/2020

L13 - RNNs

- (*) Simple RNNs generated data at beginning
BR: Raises the idea of coding constraints, ie. RNN generating output that works within constraints.
(*) Predictions are contingent on past predictions.

(*) Modelling series

- focus on sequential data
- stock example: vector of information i.r.v.s. changing day by day.
(*) Distinction between classification, prediction
(*) output may be series

(*) Representational shortcut

- use a diagrammatic representation of RNNs (different from before).
- important as we will use all throughout presentation
- A square represents series of components/neurons. (entire layer with many neurons)
- Arrows → full interconnections
- BR: Representation shows 2 hidden layers
(*) show what happens also when using 2 input vectors
- Blue squares - inputs ('into page')
- Green squares - layer of neurons
- Arrows - complete set of interconnections

(*) Stock prediction

BR: Examine all history of stock price to make current prediction.
- This is pedagogical; there are many assumptions/caveats with such an approach...
- CNN applied to series data ⟶ time delay NN.

(*) Stock predictor

- Sliding window
- There is an issue

(*) Finite response system

$y_t = f(x_t, x_{t-1}, \ldots, x_{t-N})$ N-width

- Reaction at time t is function of previous N inputs.

(AT): How can you contextualise this with respect to your existing knowledge of time series

- only 'finite window' of history considered.

(BR) frames this in terms of history

(*) Increase amount of 'history' being used to make predictions - increase window width. (N).

BR: Does this actually satisfy what we require?

(*) Long-term dependencies

- Note this is a signal processing instructor commenting on stock price movements; not econ/finance professor.
- Interesting to see their view of the problem

(W): A lot of assumptions being emailed into this example.

(*) Infinite response systems

BR: We want to look at very long term dependencies "infinite memory"
- And do this without exponential blow up of connections?

BR: What kind of feature of NN are we looking for?

(W): Memory x Feedback (crucial cybernetic principle you didn't think about)

- $y_t = f(x_t, y_{t-1})$
- Init state:- $t=0 \rightarrow y_{-1}$

$y_0 = f(x_0, y_{-1})$; $y_1 = f_1(x_1, y_0)$ ...

- Recurrence

(*) A single output influences output 'for rest of time'

(*) NARX network - nonlinear autoregressive network with exogenous input

- information about past ($x_t, x_{t-1}, \ldots$) 'stored' in output!

$$y_t = f(x_{0:t}, y_{0:t-1})$$

- output contains info about entire past

(Q): scrutinise ass. here.; how do RNNs perform against time series and econometric models?

(*) we may also want past influence to dampen influence on present also...

(*) one-tap NARX network

- NARX net with recursion from output.
- entire sequence of computation shown (unfolded) (but still one NN).
- Brown boxes show output nodes ? ?
- Yellow boxes outputs
- similarity to scanning MLP (one NN)

(*) Difference between this and previous NNs covered?

- Every column in this representation is identical

(*) Condense the colour code $\longrightarrow$ brown nodes - output.

(*) Info is 'stored' in output

(*) Generic NARX network

(*) can use arbitrary units to feed into next.

$$y_t = f(x_t, \ldots, x_{t-L}, y_{t-1}, \ldots, y_{t-K})$$

- output at t, y_t depends on past K outputs and L inputs

(*) Complete NARX network (infinite past outputs, inputs) $\longrightarrow$ not practical...

(*) NARX network

- was popular for time-series (eg. stocks, weather) for phenomena with distinct seq. of innovations.

OR: Issue: - No genuine 'memory'; a 'substitute' is the 'storage' of info in y_t output (and not in the network).

(*) Nothing in network stores information (or dedicated to storage of info.

(*) Make memory explicit

(*) Introduce a memory variable

$m_t = r(y_{t-1}, h_{t-1}, m_{t-1})$

$h_t = f(x_t, m_t)$

$y_t = g(h_t)$

m_t - memory variable (stored in memory unit)

y_{t-1} - output $(t-1)$

h_{t-1} - hidden state $(t-1)$

m_{t-1} — " —

(*) Jordan Network

(*) memory unit - dark blue ?

- introduced memory variable as described before
- introduce a running average of past output (memory unit)

(*) memory unit influenced hidden state at next time point.

(*) memory has 'fixed structure'; does not 'learn' to remember.

(*) Elman network

- separate memory state from output
- memory 'clones' previous state
- training → slices network
- treat each slice as independent predictors ~~with~~ (?)
- use slices to estimate parameters

(*) no transfer of gradients from one to another (13) - review

---

(*) there are slides in S.2020 (NARX, Elman, Jordan networks) missing from S.2019

(*) recurrence for Jordan, Elman → partial
- only occurs during operational phase; not exploited during training

---

(*) Alternate model for infinite response systems

- state space model (connection with IO-708?)

$h_t = f(x_t, h_{t-1})$

$y_t = g(h_t)$

h_t - state of network

- model directly embeds the memory in the state.

- need to define initial state h_{-1}

(*) Need to define an initial hidden state h_{-1} as a 'parameter'

(*) h_t (state) - function of current input, past state h_{t-1} (x_t)

y_t (output) - function of current state (h_t)

(14) Distinction between this and previous? (see diagrams)

(*) State at any time is function of state at previous time.

↳ information 'passed forward'

(*) Fully recurrent, if modelling f and g as NNs

functions

(*) From now on, use recurrent as qualifier

(*) Simple state space model

(*) Recurrence → cannot draw each column independently of every other column.

- information flowing through continuously.

(*) Have to show entire sequence of computations

(*) h has been used as a generic representation of a state

- no specification of what this is so far.

(*) 'State' can be arbitrarily complex

(*) Single/multi-layer RNN / multiple recurrent layer RNN

(*) Note architecture

- Distinction - state defined in a different way
- Distinction - skip connections

(*) A more complex state

- connections more complex; basic idea same

(*) essentially showing variants on same principle

(*) Or network may be even more comp!.

- no longer a standard state space model
- As there is recurrence in output - SSM - no recurrence in output

(*) Generalisation to other recurrences

- can generalise to recurrence over wider window (?)
- BR terms this 'deeper' recurrence; and I don't like the imprecision of the semantics

- ~~No~~ deeper recurrence → initial states (further back) need to be specified e.g. 2 steps back h_{-1} h_{-2} for 2-step recurrence

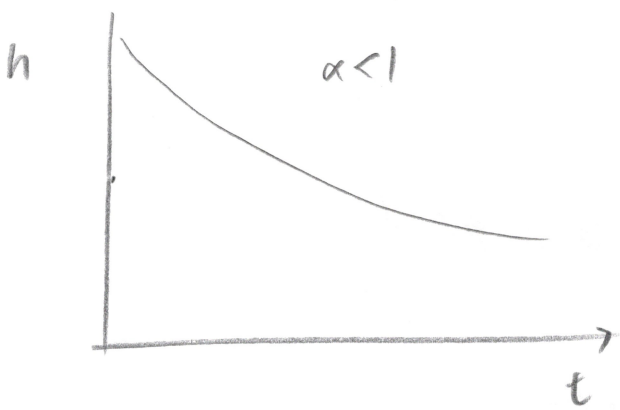
Illustrate

E.g. $h_t = \alpha h_{t-1}$

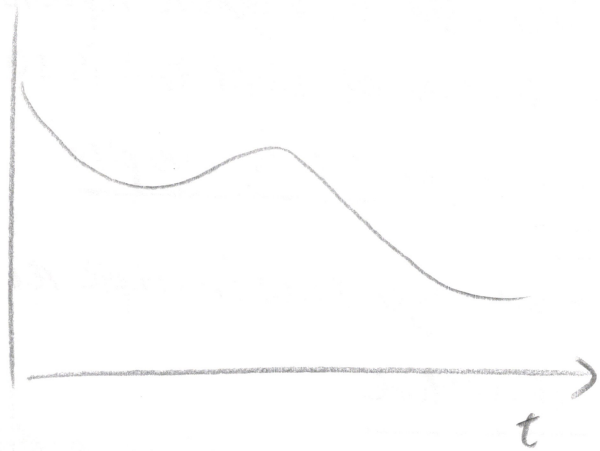
$\alpha < 1$ → decays exponentially h_t as $t \to \infty$

- Suppose you want oscillation → needs 'deeper time-steps'

$h_t = \alpha h_{t-1}$



$h_t = \alpha h_{t-1} + \beta h_{t-2}$



(*) "deeper recurrence"
can capture more complex patterns in memory

(*) simplest structures are most popular

(*) Recurrent Neural Network

(*) introduces 'loop notation' for drawing RNNs (similar to plate notation)

(*) Feedback loop/ does not refer to a violation of causality
recurrence (there is temporal unfoldment)

(*) Feedback is to save variable in next period time.

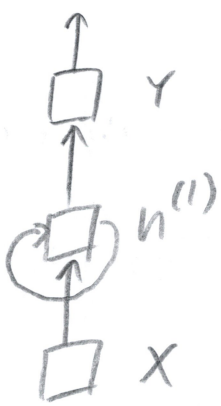
(*) Tailed/simplified

(*) Enfoldeard and unfolded representation of RNN

(*) note → squares - full layer
arrows - full connects. (e.g. K×L weight matrix)

(*) RNN equations (*)

(*):-



$n_i(-1)$ = part of net. params.

$$n_i^{(1)}(t) = f_1\left(\sum_j w_{ji}^{(1)} x_j(t) + \sum_j w_{ji}^{(11)} h_i^{(1)}(t-1) + b_i^{(1)}\right)$$

(i) (ii)

$$y(t) = f_2\left(\sum_j w_{jk}^{(2)} h_j^{(1)} + b_k^{(2)} \quad k = 1, \ldots, M\right)$$

- superscript → layer of net. from which inputs obtained
- vector fnction at output
- state node activ - $f_1(n)$ - typically tanh()
- every neuron - bias input

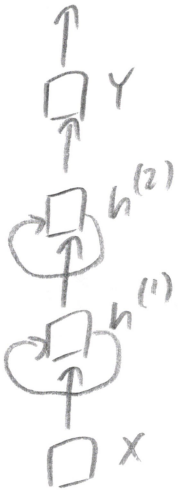
i) current weights

ii) recurrent weights.

(*) Note activation $f_1()$ acts on a weighted sum of inputs AND previous time states (with bias)

(*) $w^{(11)}$ - connection from neuron in 1st layer 'back' to neuron in 1st layer

(*) Multiple recurrent RNN



$h_i^{(1)}(t)$ = part of net params.

$h_i^{(2)}(-1)$ = —— " ——

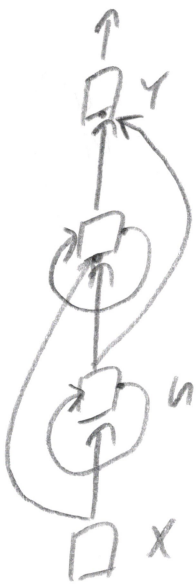
$$h_i^{(1)}(t) = f_1\left(\sum_j w_{ji}^{(0)} x_j(t) + \sum_j w_{ji}^{(11)} h_i^{(1)}(t-1) + b_i^{(1)}\right)$$

$$h_i^{(2)}(t) = f_2\left(\sum_j w_{ji}^{(1)} h_j^{(1)}(t) + \sum_j w_{ji}^{(22)} h_i^{(2)}(t-1) + b_i^{(2)}\right)$$

$$y(t) = f_3\left(\sum_j w_{jk}^{(2)} h_j^{(2)}(t) + b_k^{(3)},\ k=1,\dots,M\right)$$

- output - vector fn
- state node activ - $f_R()$ - typically tanh

(*) Multiple RNN (skip conn.)



$h_i^{(1)}(-1)$ = network param.

$h_i^{(2)}(-1)$ = —— " ——

$$h_i^{(1)}(t) = f_1\left(\sum_j w_{ji}^{(0,1)} x_j(t) + \sum_i w_{ii}^{(1,1)} h_i^{(1)}(t-1) + b_i^{(1)}\right)$$

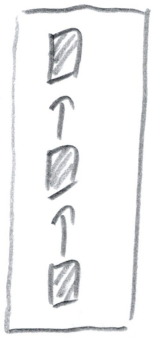
$$h_i^{(2)}(t) = f_2\left(\sum_j w_{ji}^{(1,2)} h_j^{(1)}(t) + \sum_j w_{ji}^{(0,2)} x_j(t) + \sum_i w_{ii}^{(2,2)} h_i^{(2)}(t-1) + b_i^{(2)}\right)$$

$$y_i(t) = f_3\left(\sum_j w_{jk}^{(2)} h_j^{(2)}(t) + \sum_j w_{jk}^{(1,3)} h_j^{(1)}(t) + b_k^{(3)}\right)\ k=1,\dots,M$$

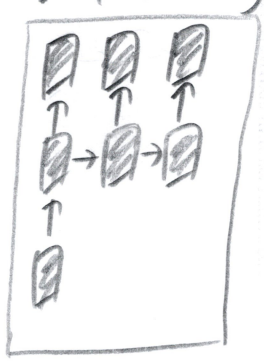
(*) Variants on RNN:-

1. Conventional MLP/NN
2. sequence generation (single input, multiple outputs)
3. sequence prediction/class-

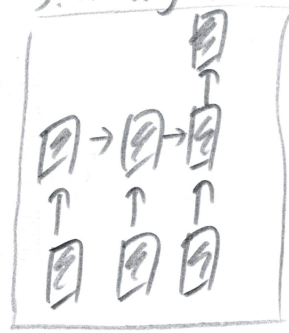
1. 1-to-1



2. 1-to-many



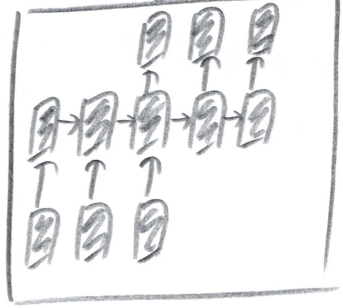
3. many-to-1



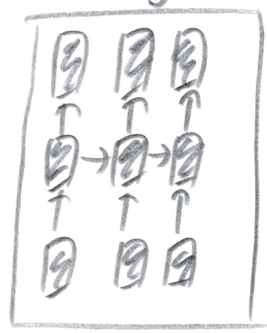
(*) variants.

1. delayed sequence
2. sequence to sequence

1. many-to-many



2. many to many



(*) story so far

(w): make sure you appreciate distinctions between NARX, 'simple' RNNs, SSMs.

(*) How to train network

- use BPTT - backprop through time.
- given a collection of sequence inputs

- $(\underline{X}_i, \underline{D}_i)$ where $\underline{X}_i = X_{i,0}, \ldots, X_{i,T}$

$\underline{D}_i = D_{i,0}, \ldots, D_{i,T}$

(*) train network params to minimise error between output of network

$\underline{Y}_i = Y_{i,0}, \ldots, Y_{i,T}$ and desired outputs

(*) assume synchronous output for every input.

(*) At the i^{th} input consists of T input vectors (for each timestep) $X_{i,t}$

i^{th} output —"— T output vectors —"— $Y_{i,t}$

(*) find a way of training/estimating params to do this.

(*) Forward pass - RNN - asyn.

- fairly trivial

(AS) - Review pseudocode (quickly).

(*) BPTT

(*) Divergence is between entire sequence of outputs produced on forward pass and entire sequence of desired outputs.

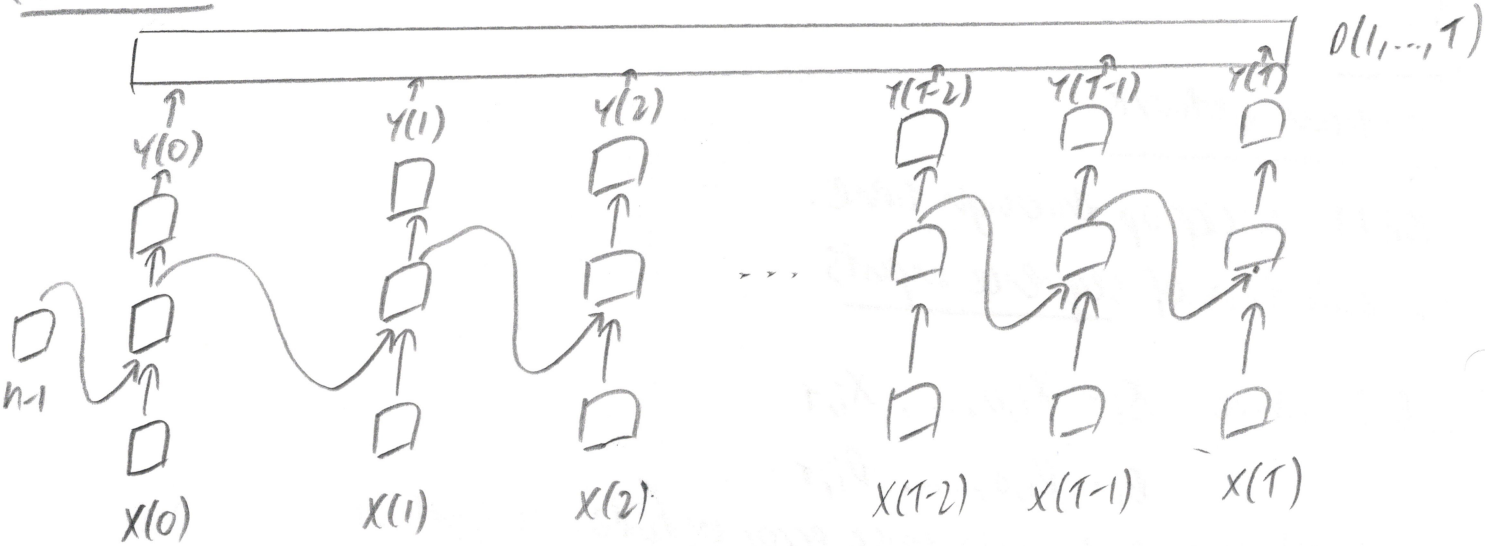
(CM) (SM)

(*) Point is there is no reason to expect an additive relationship between individual divergence at each time point; and divergence over whole sequence.

(*) Due to alignments/subst. etc.

(*) But can be explicitly assumed in special cases.

(*) Notation



(one hidden layer example)

- $Y(t)$ - output at time t
  - $Y_i(t)$ - ith output
- $z^{(2)}(t)$ - pre-activation value of neurons at output layer at time t
- $h(t)$ - output of hidden layer at time t
- $z^{(1)}(t)$ - pre-activation value of hidden layer at time t

(AS) - Difference in subscript between S.2019/S.2020 slides.

(*) BPTT

(*) Note divergence is a vector function of output

(*) Have to compute in totality

- But do not confuse divergence being a vector fn with being able to take derivatives wrt a component of output (?)

- DIV is a function of $Y(0), Y(1), \ldots Y(T)$

Ouse 'DIV' vs 'div' in previous to emphasise diff

(*) compute $\frac{dDIV}{dY_i(t)}$ $\forall i, t$ (i.e. over all $i=1,\ldots,N$ observation sequences, $t=1,\ldots T$ time steps)

(*) Can be a source of difficulty;

(*) BPTT

(*) explicitly assume that :-

overall divergence is an additive sum of local divergences at each time.

(*) Require: $\frac{dDIV}{dY_i(t)}$ $\forall i, t$

(*) will usually get $\frac{dDIV}{dy_i(t)} = \frac{dDiv(t)}{dy_i(t)}$ (?)

(*) Assumed for pedagogical purposes; in practice, BPTT is not contingent on this assumption.

(AZ). Note picture showing this

(*) BPTT (equations) (for one training instance)

BR: uses this convention

- compute $\frac{dDiv}{dY_i(T)}$ for all i

(AE): make sure these are crystal clear

- Assuming Y is column vector; $\nabla_{Y_t} Div$ is a row vector

- Conventional backprop

(*) $\nabla_{Y_T} DIV = [$ ——— $]$

$\nabla_{z^{(1)}(T)} DIV = [$ ——— $] =$ 

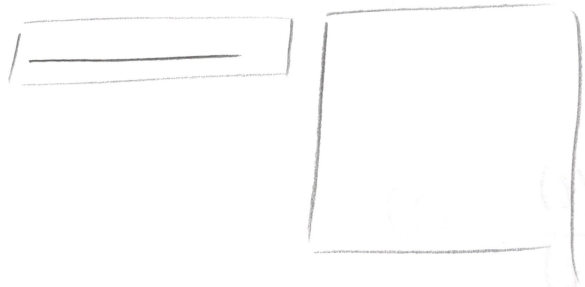
(*) $$\nabla_{z^{(2)}(T)} DIV = \nabla_{Y(T)} DIV \nabla_{z^{(2)}(T)} Y(T) = \nabla_{Y(T)} DIV \; J_{z^{(2)}(T)}(Y(T))$$

(*) Note in scalar form:-

$$\underbrace{\frac{dDIV}{dz_i^{(2)}(T)} = \frac{dDIV}{dY_i(T)} \frac{dY_i(T)}{dz_i^{(2)}(T)}}_{\text{scalar activation}} \quad OR \quad \underbrace{\frac{dDIV}{dz_i^{(2)}(T)} = \sum_j \frac{dDIV}{dY_j(T)} \frac{dY_j(T)}{dz_j^{(2)}(T)}}_{\text{vector activation.}}$$

$$\nabla_{h(T)} DIV = \nabla_{z^{(2)}(T)} DIV \; \nabla_{h(T)} z^{(2)}(T) \qquad \nabla_{h(T)} DIV \text{ - row vector}$$

$$= \nabla_{z^{(2)}(T)} DIV \; W^{(2)}$$



Scalar form: $$\frac{dDIV}{dh_i(T)} = \sum_j \frac{dDIV}{dz_j^{(2)}(T)} \frac{dz_j^{(2)}(T)}{dh_i(T)} = \sum_j w_{ij}^{(2)} \frac{dDIV}{dz_j^{(2)}(T)}$$

$$\nabla_{W^{(2)}} DIV = \nabla_{W^{(2)}} z^{(2)}(T) \nabla_{z^{(2)}(T)} DIV$$

$$= h(T) \nabla_{z^{(2)}(T)} DIV$$

Scalar:- $$\frac{dDIV}{dw_{ij}^{(2)}} = \frac{dDIV}{dz_j^{(2)}(T)} h_i(T)$$

(*)
$$\nabla_{z^{(1)}(T)} DIV = \nabla_{h(T)} DIV \nabla_{z^{(1)}(T)} h(T)$$

scalar form:-
$$\frac{dDIV}{dz_i^{(1)}(T)} = \frac{dDIV}{dh_i(T)} \frac{dh_i(T)}{dz_i^{(1)}(T)}$$

(*)
$$\nabla_{W^{(1)}} DIV = \nabla_{W^{(1)}} z^{(1)}(T) \nabla_{z^{(1)}(T)} DIV$$

$$= x(T) \nabla_{z^{(1)}(T)} DIV$$

scal:
$$\frac{dDIV}{dw_{ij}^{(1)}} = \frac{dDIV}{z_j^{(1)}(T)} x_i(T)$$

(*)
$$\nabla_{W^{(1,1)}} DIV = \nabla_{W^{(1,1)}} z^{(1)}(T) \nabla_{z^{(1)}(T)} DIV$$ (gradient wrt recurrent weight)

$$= h(T-1) \nabla_{z^{(1)}(T)} DIV$$

scal:-
$$\frac{dDIV}{dw_{ij}^{(1,1)}} = \frac{dDIV}{dz_j^{(1)}(T)} h_i(T-1)$$

(*). (OTT): At this stage, we have only gone back one time step

- difference between previous presentations of conventional (MLP) backprop)

CNN backprop:-
- Have to backprop all the way (to be clarified)

(*) omitting scalar form from hereon:- → S.2020 typo z

(*) (OTT)
$$\nabla_{z^{(2)}(T-1)} DIV = \nabla_{y(T-1)} DIV \nabla_{z^{(2)}(T-1)} y(T-1)$$

(analog to earlier, scalar form for both vector act/ convent.)

(*) $\nabla_{h(T-1)} DIV = \nabla_{z^{(2)}(T-1)} DIV \nabla_{h(T-1)} z^{(2)}(T-1) + \nabla_{z^{(1)}(T)} DIV \nabla_{h(T-1)} z^{(1)}(T)$

$= \nabla_{z^{(2)}(T-1)} DIV \, W^{(2)} + \nabla_{z^{(1)}(T)} DIV \, W^{(1,1)}$

scalar form: $\frac{dDIV}{dh_i(T-1)} = \sum_j w_{ij}^{(2)} \frac{dDIV}{dz_j^{(2)}(T-1)} + \sum_j w_{ij}^{(1,1)} \frac{dDIV}{dz_j^{(1)}(T)}$

(51): note this has the above functional form :- examine route of influence from $h(T-1)$ to DIV (through 2 distinct routes)

(49): view with slides (!) - v. important

(50): - $\nabla_{W^{(2)}} DIV \mathrel{+}= \nabla_{W^{(2)}} z^{(2)}(T-1) \nabla_{z^{(2)}(T-1)} DIV$

$\Rightarrow$ (*) $\nabla_{W^{(2)}} DIV \mathrel{+}= h(T-1) \nabla_{z^{(2)}(T-1)} DIV$

scalar form: $\frac{dDIV}{dw_{ij}^{(2)}} \mathrel{+}= \frac{dDIV}{dz_j^{(2)}(T-1)} h_i(T-1)$

(51): computing $\nabla_{W^{(2)}} DIV$ requires summation / update as the weight is identical to what was computed previously at time step T

$\left(\nabla_{W^{(2)}} DIV = h(T) \nabla_{z^{(2)}(T)} DIV \right)$

(51): This is a significant change.

- similar to before :- $\nabla_{z^{(1)}(T-1)} DIV = \nabla_{h(T-1)} DIV \nabla_{z^{(1)}(T-1)} h(T-1)$ (same logic)

scalar form:- $\frac{dDIV}{dz_i^{(1)}(T-1)} = \frac{dDIV}{dh_i(T-1)} \frac{dh_i(T-1)}{dz_i^{(1)}(T-1)}$

(III): similar principle of 'inputting' applies as (II), :-

(*) $\nabla_{W^{(1)}} DIV \mathrel{+}= \nabla_{W^{(1)}} z^{(1)}(T-1) \nabla_{z^{(1)}(T-1)} DIV$

$\Rightarrow \nabla_{W^{(1)}} DIV \mathrel{+}= X(T-1) \nabla_{z^{(1)}(T-1)} DIV$

$\frac{dDIV}{dw_{ij}^{(1)}} \mathrel{+}= \frac{dDIV}{dz_j^{(1)}(T-1)} x_i(T-1)$

(*) And also for recurrent weights:-

$\nabla_{W^{(1,1)}} DIV \mathrel{+}= \nabla_{W^{(1,1)}} z^{(1)}(T-1) \nabla_{z^{(1)}(T-1)} DIV$

$\Rightarrow$

(*) $\nabla_{W^{(1,1)}} DIV \mathrel{+}= h(T-2) \nabla_{z^{(1)}(T-1)} DIV$

scalar form:- $\frac{dDIV}{dw_{ij}^{(1,1)}} \mathrel{+}= \frac{dDIV}{dz_j^{(1)}(T-1)} h_i(T-2)$

(...)

- keep iterating using all principles above until reach initial state h_{-1}

(III) h_{-1} is a param. equiv.

$\nabla_{h_{-1}} DIV = \nabla_{z^{(1)}(0)} DIV \nabla_{h_{-1}} z^{(1)} 0$

$= \nabla_{z^{(1)}(0)} DIV \, W^{(1,1)}$

$\frac{dDIV}{dh_i(-1)} = \sum_j w_{ij}^{(1,1)} \frac{dDIV}{dz_j^{(1)}(0)}$

(#) BPTT pseudocode

(A10) - Review

- why are there many t= → resolve
- which ones are incrementing; and which aren't?

(?) AReview why ~~t-3~~ n derivative of n is incremented (?) (?)

(*) Extensions to RNN: Bidirectional RNN (BRNNs)

- RNN with forward and backward recursion (?) (A11)
  - explicitly models the fact that just as the future can be predicted from the past; past can be deduced from future (???)
- Read paper
- use future output to make inferences

(*) BRNN

- forward net processes input from t=0 ··· t=T
- Backward ——— " ——— t=T, ..., t=0

(*) BRNN: Processing input (?)

- forward pass } through BRNN produces hidden state rep of input.
- backward pass } ^2 ^ ^ represented

(!) note this is not backprop!

(*) output uses both 'hidden state representations'

(*) BRNN - pseudocode

(A11) - Review

- (B11): forward pass
  back pass

(Q11): Affine combination of output is weighted combination of forward pass with own set of weights, and backward pass (with own set of weights) and bias

-uses hidden state rep from forward and backward pass

(*) Bidirectional - simplified code

(*~~+n~~ (A12): clever flipping to implement - review.

(*) Backprop in BRNNs

-see slides

forward

(Q11): Propagate inputs through forward network, backward network.

↑ to get final output; compute divergences

(Q1) Backprop: -

1) Forward net - ignore backward network; perform BPTT to get derivatives for the forward network; update parameters.

2) Backward net - ignore forward network; perform BPTT to get derivatives for back network; update params.

(*) Backprop for BRNNs

(A13) - review pseudocode

2 step: -

i) BPTT subroutine

ii) Param updates
