11-785 07/07/2020

L14 - RNNs: stability analysis LSTMs

Review

- Heated structures - short term dependences
- Recurrent — " — long term — " —
(*) This is not entirely true

(*) Recurrent structures vs static structures

(Q1). Strange how symbolic ops modelled in MLPs

- Addition problem - 2 N bit numbers
- 'Learn' the operation
- Network will be large (exponential)
- Training data will be exponentially large (amt required)

(*) MLPs vs RNNs

- Looks as recurrent problem advantages
- RNN: could compute using few neurons what a fixed/static structure would take an exponential no. to compute
(+) (Q7): It's not merely problems with long term dependencies that are benefitted by RNN style architecture
- MLP with RNN architect can be more efficient by in terms of network param, amount of training data

---

(*) Types of recursion

(*) Behaviour of recurrence

- Finite-response / time delay
- If input is always bounded; what is condition for input to blow up?

(Q2): If output is a multiple of input that is greater than 1 (so input at, $X_{t+1} = KX_t$ $K>1$)

(*) BR: will never 'explode' (?)
- unless activations have singularities, output will remain bounded if input is bounded

} (Q1) - review notes

(*) Time delay structure
- Bounded input, bounded output stability BIBO stability (P2) SP team

(*) Is this BIBO?

(?): what are conditions under which output will not explode?
If hidden activations and outputs are bounded

(?): will it saturate and where? - saturate → meaningless results.

what if activations linear?

BIBOstability: output carries info about input in a way that there isn't saturation

(*) Analysing recurrence
- output depends only on current hidden activation
- If you want to know how output will behave → analyse hidden layer h_R
(*) Examine linear activations

(*) streetlight effect
- analyse linear systems; extrapolate to non-linear systems

- $z_R = W_h h_{R-1} + W_x x_R \qquad h_R = z_R$

(*) linear systems
- unfold the recurrence
(P2): Review - expand recursion

$$h_R = W_h^{R+1} h_{-1} + W_h^{R} W_x x_0 + W_h^{R-1} W_x x_1 + W_h^{R-2} W_x x_2 + \dots$$
(·) ↙

$$h_R = H_R(h_{-1}) + H_R(x_0) + H_R(x_1) + H_R(x_2) + \dots$$

(·): - response to an input x_0 at time 0; when there are no other inputs, zero initial condition

↙: rewrite operations: $H_R(x_0)$ - response at time R assuming input x_0 at time 0, and all other inputs 0. (deterministic)

(*) As system is linear, can consider x_0 as scaling factor, pull it out yielding:-

$h_R = h_{-1} H_R(l_{-1}) + x_0 H_R(l_0) + \dots + x_2 H_R(l_2) + \dots$

(*) For linear activation, responses at time R can be thought of as summations of individual inputs, assuming all else 0.

(*) Examine what happens given network receives single input at time 0.

- If that blows up ⟶ system blows up
- If that does not blow up ⟶ — " — not blow up
- principle of superposition in linear system (A4) review

(*) Linear recursion

- scalar recurrence

$h(t) = wh(t-1) + cx(t)$

$h_0(t) = w^t cx(0)$

- subscript 0 means input occurred at time 0!

- Behavior of $h_0(t)$?

(W): similar to argument previously; $|w|>1$ $t>1$ ✓✓

- stability is only when $w=1$.

(*) Linear recursion: vector recursion

$h(t) = Wh(t-1) + Cx(t)$

$h_0(t) = W^t Cx(0)$

- use eigendecomposition of W (weights matrix). $W = U\Lambda U^{-1}$

BR: columns of U? (eigenvectors)

(W): - orthonormal (basis vectors)
- linearly independent

- note $Wu_i = \lambda u_i$

- for any vector $x' = cx$ ← (A5) slides S.2020 → print

$x' = a_1u_1 + a_2u_2 + \dots + a_nu_n$

$Wx' = Wa_1u_1 + Wa_2u_2 + \dots + Wa_nu_n$

$= a_1Wu_1 + a_2Wu_2 + \dots + a_nWu_n$ (as a_n is scalar)

$= \lambda_1u_1 \quad = \lambda_2u_2$

$Wx' = a_1\lambda_1u_1 + a_2\lambda_2u_2 + \dots + a_n\lambda_nu_n$

$W^tx' = a_1\lambda_1^tu_1 + \dots + a_n\lambda_n^tu_n$

(*) consider behaviour as $t \to \infty$

$\lim_{t\to\infty} |W^tx'| = a_n\lambda_m^tu_m = c\max_j \lambda_j$

(A6): BR's arguments are a little unclear to me rn
(convergence)

(?)
BR: ability of network to 'remember' depends on largest eigenvalue of W.

(*): 'All of other a_is' disappeared
- (?) Not sure what is meant by length of hidden vector will expand/contract

(?): $|\lambda_{max}| > 1$? - determines blow up

(*) complex eigenvalues → oscillation + blow up.

---

lesson...

(BR): linear systems: -
- long term behaviour depends on eigenvalues of the hidden weights matrix. $|\lambda_{max}| > 1$
- largest eigenvalue > 1 → system blows up
- ——"—— $|\lambda_{max}| < 1$ → response will vanish quickly
- complex eigenvalues cause oscillatory behaviour on top of divergence/blow-up
- smooth behaviour → force weights matrix to have real eigenvalues (A6)
  - enforce symmetric weights matrix

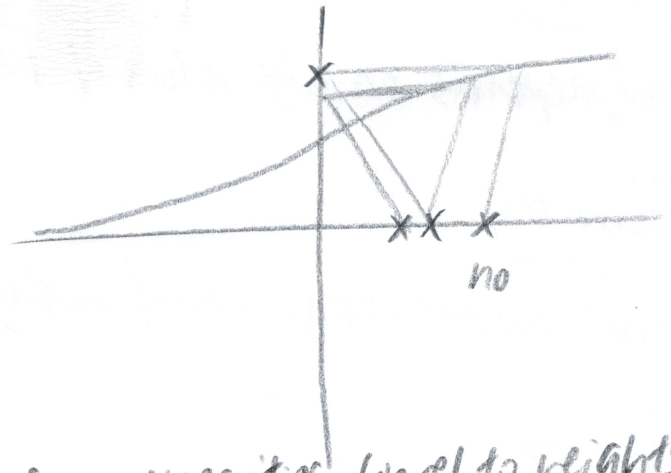
(*) How about non-linearities (scalar)

- consider :- $n(t) = f(wh(t-1) + cx(t))$

$h_t = \sigma(wh_{t-1} + x)$

OR: what is largest slope of sigmoid function ?

- "w or 1"
- assume it's 1
- what happens when initial value at h_0



(**): some key insights on behaviour of non-linearities (in rel to weights, biases)

- sigmoid, tanh, ReLU
- diagrams on slide ① and ② differ by whether initial value h_0 is to the left of 0 (<0) or >0 on the plot of ~~input~~ output vs input on diagram above.

(*) vector Process

- analysis more complex
- $n(t) = f(wh(t-1) + cx(t))$
- uniform unit vector init: $[1, 1, \ldots] / \sqrt{N}$

(*) Behaviour similar to scalar recursion

(*) Stability Analysis

- need more specialised tools for considerations of stability formally i.e. Lyapunov stability, Routh's criterion

(*) (**): Comments on saturation profiles:- (activations all eventually saturate)

i) sigmoid: saturate quickly

ii) tanh: retain memory about where you started from (initial cond?) for a large period of time (but also eventually saturate)

(*) ReLUs → not used (blow up)

(*) (07): Activations in RNNs tend to be tanh - retain 'memory' for long enough

(*) Behaviour also depends on eigenvalues of W.

(*) Deeper recursion

BR: conclusions do not change significantly

(*) Story so far (07)

- Excellent summary

(*) Vanishing gradients

(*) Issue with any deep network

- Gradient of error/loss at beginning of network is 'unstable'
- Why?
- (07): Note nested function rep. of NN
- MLP: $Y = f_N(W_{N-1} f_{N-1}(W_{N-2} f_{N-2}(\dots(W_0 X))\dots))$

- W_k - weights matrix in k^{th} layer

$$\text{Error/Div} = \text{Div}(X) = D(f_N(W_{N-1} f_{N-1}(W_{N-2} f_{N-2}(\dots(W_0 X)\dots))$$

(*) Training deep nets

- How to find $\nabla_X f$ where $f = f(Wg(X))$
- $\nabla_Z f$ - Jacobian matrix of $f(z)$ wrt z - i.e. $J_z(f)$
- $z = Wg(X)$

$$\nabla_X f = \nabla_Z f \cdot \nabla_g z \cdot \nabla_X g = \nabla_Z f \cdot W \cdot \nabla_X g$$

For

$Div(X) = D(f_N(W_{N-1} f_{N-1}(W_{N-2} f_{N-2}(\dots(W_0 X))\dots))$ (should be $Div(X, Y)$)

- Y suppressed

$\nabla_{f_k} Div = \nabla D . \nabla f_N . W_{N-1} . \nabla f_{N-1} . W_{N-2} \dots \nabla f_{k+1} W_k$

- $\nabla_{f_k} Div$ - gradient of error $Div(X)$ wrt output of k^{th} layer of network
  - required to compute $\nabla_{W_{k-1}} Div$ and $\nabla_{b_{k-1}} Div$.

- ∇_{f_n} is Jacobian of $f_N()$ wrt <u>current input</u>

- clarity:-

(*) <u>Jacobian of hidden layers (RNN)</u>

$h_i^{(t)}(t) = f_i(z_i^{(t)}(t))$

$$\nabla f_t(z_t) = \begin{bmatrix} f'_{t,1}(z_1) & 0 & \dots & 0 \\ \vdots & f'_{t,2}(z_2) & & \vdots \\ & & \ddots & 0 \\ 0 & 0 & \dots & f'_{t,N}(z_N) \end{bmatrix}$$

(*) Derivative (subgradient) of activation is always bounded.

- diagonals/singular values of Jacob - bounded.

(*) Limit on how much multiplying a vector by Jacobian will scale it.

<u>Notation:</u> $\nabla f_t()$ is the derivative of output of the (layer of) hidden recurrent neurons with respect to their input

<u>S. 2019:-</u>

- vector activations - full matrix
- scalar activations - matrix where diagonal entries are derivatives of hidden layer activations

(A7) - unclear about indexing

- derivative of hidden state act.
- most common activation fns: sigmoid(), tanh, RELU have derivatives less than 1
- most common RNN act (RELU) - tanh()
  - derivative of tanh() is never greater than 1

(A8) - check on Wolfram

(A7): multiplication by Jacobian → shrinking operation

(A7): Backprop → Jacobians shrink the derivative.

(*) What about weights

(A7): single layer RNN - identical weight matrices (A9) ? singular values?

cross product ∇_{fx} Div:

i) Expands ∇D along directions in which singular values of weight matrices are greater than 1

ii) shrinks ∇D in directions where singular values are less than 1

- does he mean eigenvalues of W? (rather than singular values?) - SVD eigenvalues.

(*) Hence exploding/vanishing gradients

(*) Gradient problems in deep nets.

- Hence gradients of div/error/loss wrt to earlie layers in network (i.e. further 'back' in backprop, from greater no. of multiplications) can explode/vanish.
- Has effect on gradient descent updates, ↑ problem with net. depth.

(*) vanishing gradient examples

QR: At what stage should derivatives be largest?

(A): Am guessing he means derivatives of loss wrt parameters

- If so; will be largest at beginning of training ✓

(*) Illustration of vanishing gradients

- true for generic neural networks

(*) Story so far

- key points - review

(*) Recurrent nets are deep nets

- even though vanishing/exploding gradients apply to deep nets (rather than RNNs specifically); they apply to RNNs if you consider these as an instance of deep nets

OR: we want RNNs to have long memory (hindered by vanishing/exp. gradient problems).

(*) Can happen on forward pass too.

(*) long-term dependency problem

- we want a way of 'remembering'; but we are in a situation where the extent of this (i.e. lack of saturation) depends on (eigenvalues of) network params (rather than inputs)

↳ LSTM - mystery/magic?

(*) Exploding/vanishing gradients

- Recall: exploding/vanishing gradients is antithetical to 'memory retention'

(*) Exploding/vanishing gradients:-

i) Behaviour of Jacobians ∇_{f_n} wrt current input

(Bounds on Jacob) depends on nature of activation

ii) Eigenvalues of weight matrices W_k

- not related to input

(OR): whether network 'remembers' depends on parameters (W_i) rather than what it is 'trying to remember' - input.

(*) looking for a different design (a 'failing' of RNN).
- input-based determination of 'memory'.

(*) use example of curly brace in sentence being remembered

(A5): design choice/aspiration:-
seek a design/architecture that:-
i) doesn't cause exploding/vanishing gradients
ii) 'retains' useful memory arbitrarily long (should know what to remember based on input)

- memory(K) ≈ $C(x).\sigma_K(x).\sigma_{K-1}(x) \ldots \sigma_1(x)$
- $\nabla_{far} Div \approx \nabla DC\sigma_N' C\sigma_{N-1}' \ldots C\sigma_K'$

- info extracted from input
- determination of what to remember - based on input based flag (by a function of input)
- no weights, no activations; only triggers that bump up/down memory.

(*) constant error carousel

(A8) - review architecture

- history stored in variable $C(t)$
- no weights, no non-linearities
(*) non linearities done by other portions of net.
$\sigma(\cdot)$ is the gate : depends on current input, hidden state (and other stuff)

- curly brace example:- (e.g. forget open brace)
i) seen closed brace - current state?
ii) know open brace was been detected and stored. - hidden state.

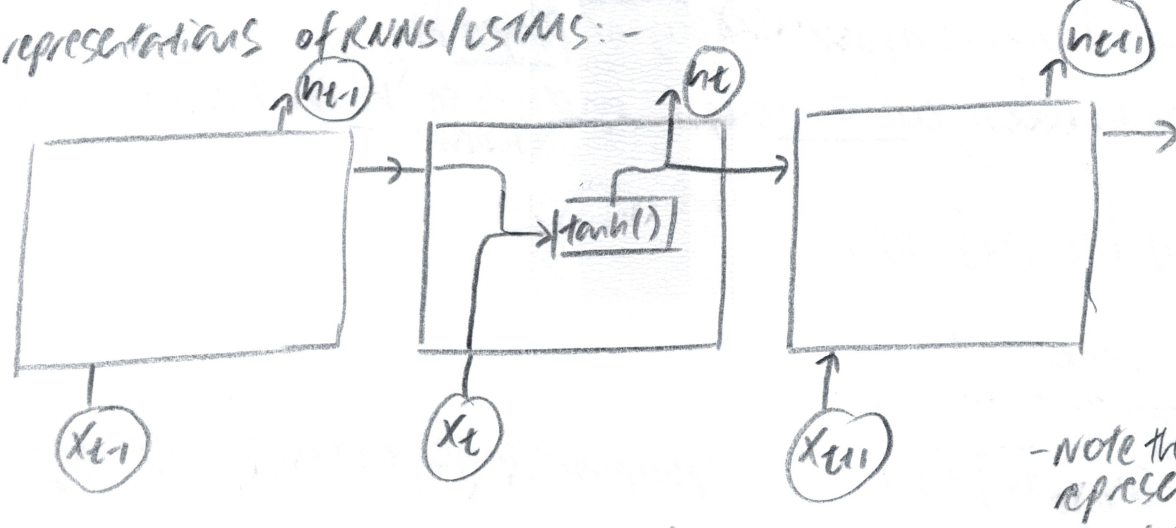
(A9) - issue with indices in S.2020 slides of LEC diagram.

(*) enter LSTM

(*) the key point is that it is an input-based determ.

(*) Note representations of RNNs/LSTMs:-

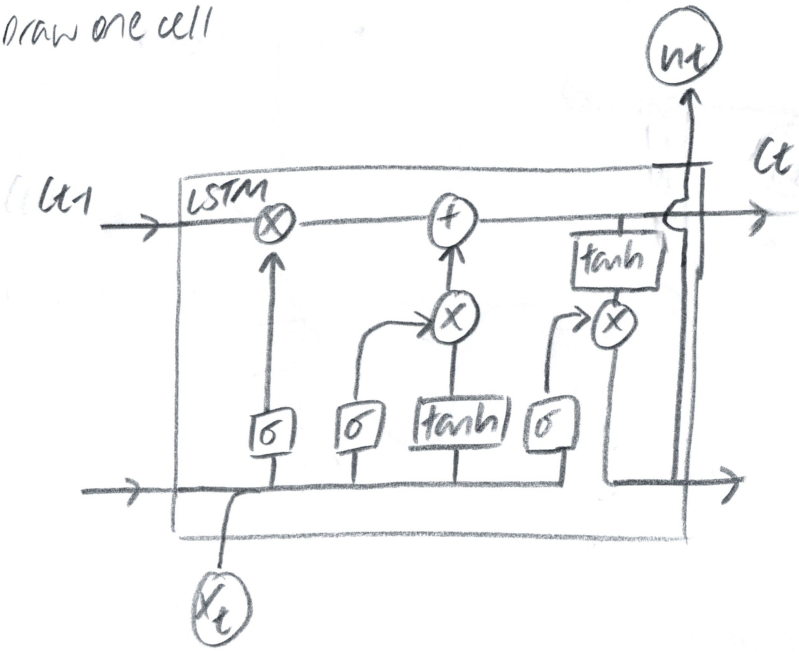
RNN



- Note the representational convention established at beginning.

- LSTMs:- Quite complex 'circuit diagrams'
- Draw/pick up in review (A10)

LSTM :- Draw one cell



BR: breaks each element/functional sub-circuit of this down

- LSTM CEC: C_t - linear history carried by CEC
  - carries info through, only affected by gate
  - And addition of history (which is gated)
- LSTM: Gates
  - Gates are simple sigmoid with output range (0,1)
  - controls how much info to be let through.

LSTM: Forget gates

- determines whether to carry our history or forget it

(*) distinction between cell memory C and state h in that is carrying ov. the hidden-

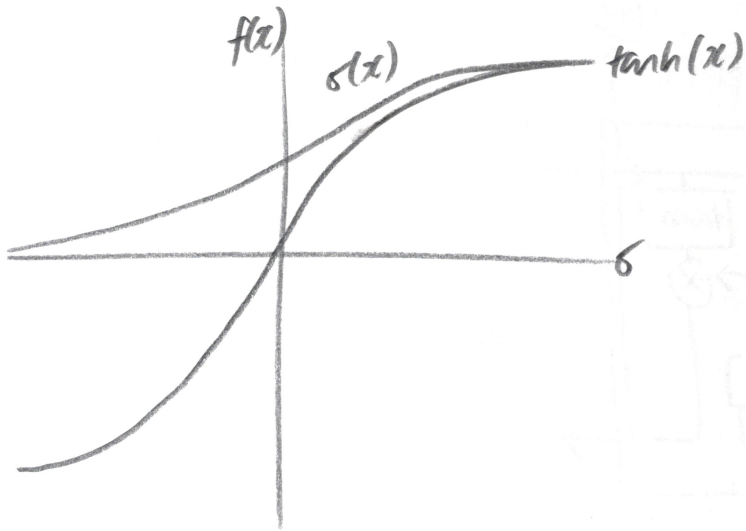
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

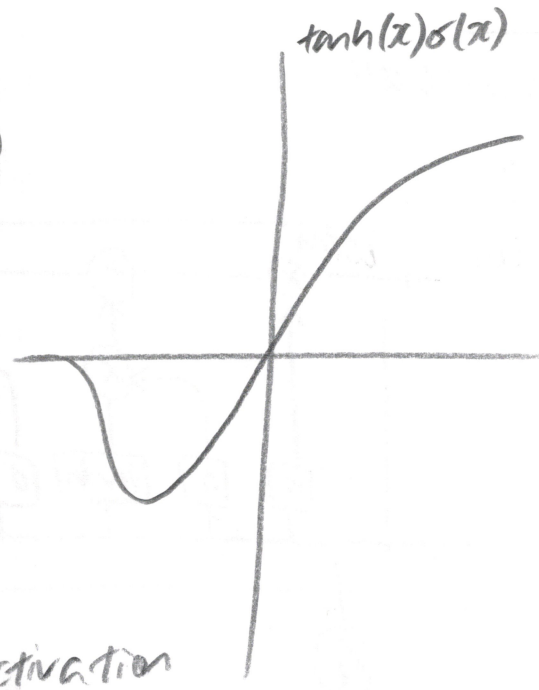
(*) LSTM: input gate

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$ - 'perceptron' layer that det. if sth new/int in input.

$$\hat{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$ - a gate that det. ...

- product of σ and tanh (math. view):-





(*) can also view as more complicated activation

(to if going for mathemat. view.

(*) Memory cell update

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

(*) previous memory (C_{t-1}) multiplied by output of forget gate f_t, added to product of input gate and pattern detector as input.

(?) which is it and $\hat{C}_t$ (A11)

LSTM: Output and output gate

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

- current memory C_t goes through, processed by tanh() to computed hidden value h_t
- hidden state h_t also multiplied by a gate that 'decides' if memory contents are worth remembering

(SH): make a distinction between intuition/expl./narra; and maths.

LSTM: Peephole connection

(see slides)

- raw memory unproductive by itself; can be input.

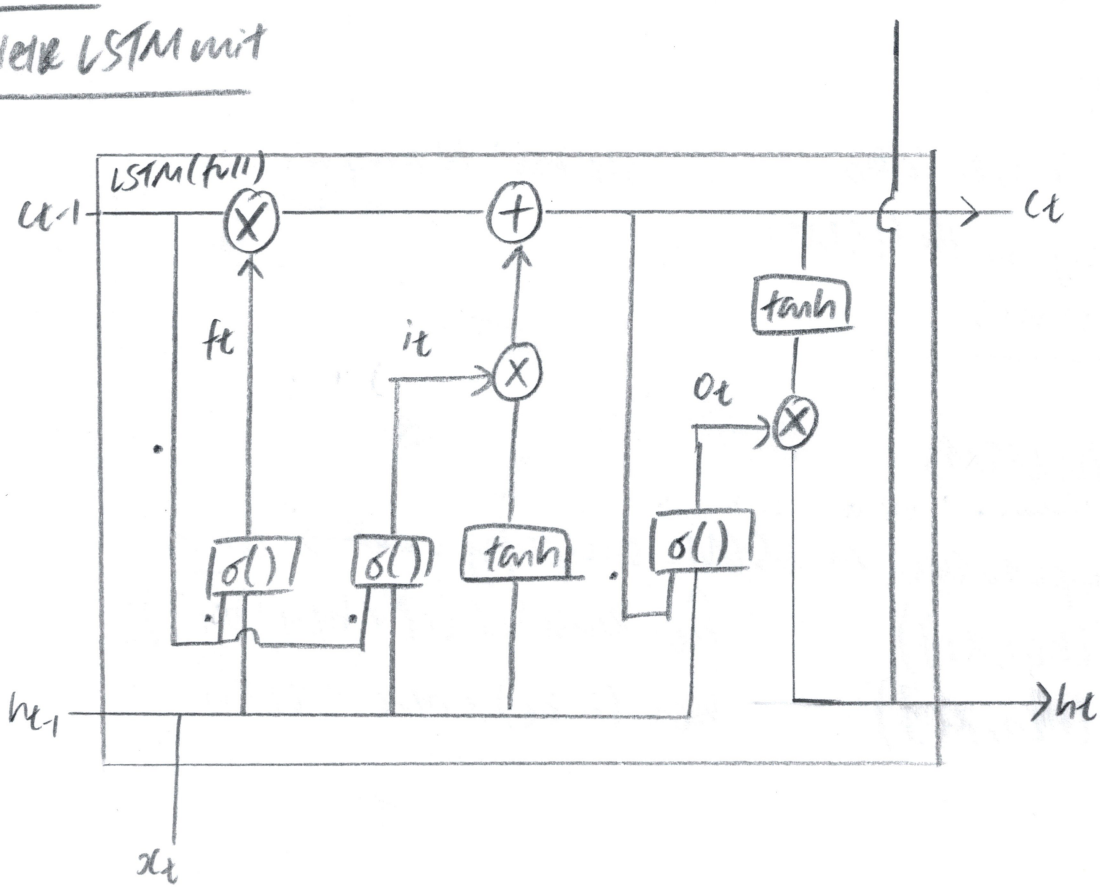
$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)$$

(·) indicates on diagram peephole conn.

(*) Complete LSTM unit



Forward rules

GATES

$f_t = \sigma(W_f \cdot [c_{t-1}, h_{t-1}, x_t] + b_f)$

$i_t = \sigma(W_i \cdot [c_{t-1}, h_{t-1}, x_t] + b_i)$

$o_t = \sigma(W_o \cdot [c_t, h_{t-1}, x_t] + b_o)$

VARIABLES

$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$

$c_t = f_t * c_{t-1} + i_t * \tilde{c}_{t-1}$

$h_t = o_t * \tanh(c_t)$

(*) LSTM forward pseudocode

- (A12) - review pseudocode (trivial if you have equations mastered)

(*) Backprop rules: Backward

BR: Head can start spinning when doing this.

(A11): BR seems to recommend foregoing maths! And instead view directly in terms of code

- new to me.

(*) LSTM backward

- assume already have dh_0
- see how to compute a deriv.

(A13) Review backprop with (LSTM) suppl. slides

(*) GRUs: Simplify LSTMs

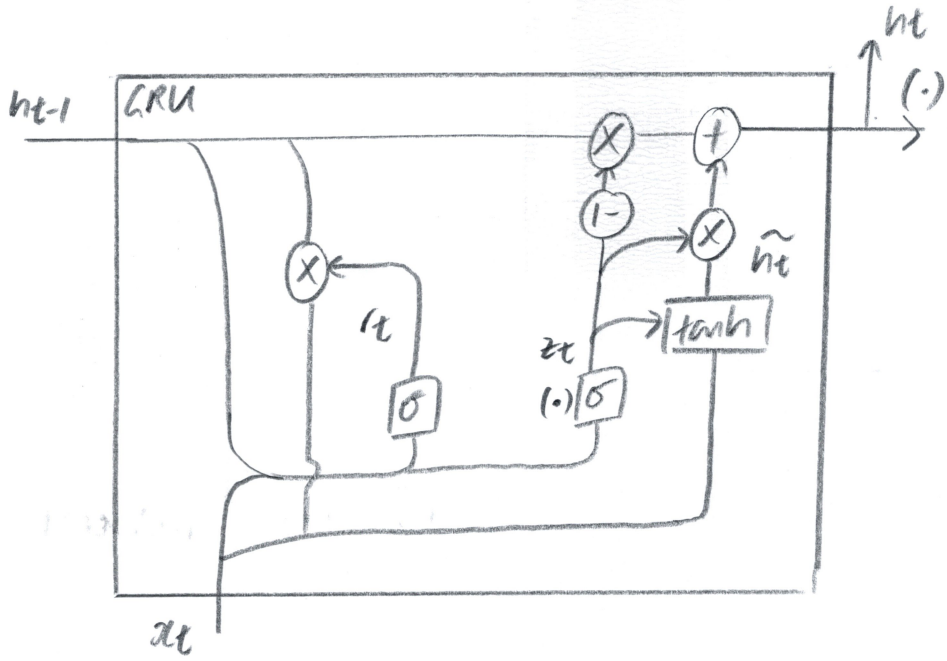
- motivitous computation in LSTM: simplify via → GRU

$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$

$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$

$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$

$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$



(·) combine forget and input gates
If a new input is to be remembered, old memory forgotten

(·) don't separately store compressed, regular memories.

LSTM eq.

(A14). review

LSTM arch, Bidirect. LSTM

(A15) - review.