Zhang - Modern RNNs

1. LSTM
2. GRU
3. Deep RNNs
4. Bi-Directional RNNs
5. Machine translation & the dataset
6. encoder - Decoder architecture
7. Seq - to - Seq
8. Beam search

the most popular RNN architectures feature mechanisms to handle the numerical instabilities faced by RNNs

↳ vanishing & exploding gradients

TWO MOST SUCCESSFUL RNN archs:

① LSTM
② Bidirectional

LSTM

has a lighter weight version called GRU

- introduces a memory cell
- A unit of computation
- replaces hard nodes in the hidden layer
- Avoids van grads by keeping values in each memory cell cascading along a recurrent edge w/ weight 1 across many time steps
- A set of gates help the network to determine what inputs to allow into the memory state
- Also, what content of memory state should influence the model output

Bidirectional

Arch where info from both past & future is used to determin output @ any point in the sequence

10.1 long Short-term Memory

- Since intro of (1980) RNNs trained using Backprop problems of long-term dependancies have been a problem

Gradient clipping helps exploding grads but not vanishing

one of first & most succ in handling vanishing = LSTM

- ↳ ordinary recurrent nodes are replaced w/ memory cells

each memory cell contains an internal state

↳ A node w/ a self connected recurrent edge of fixed weight 1

ensures gradient can pass many time steps w/out vanishing or exploding

the name LSTM comes from the following:

Simple RNNs have long-term memory in the form of weights

↳ the weights can change slowly during training encoding general knowlege about the data

the also have short-term memory in the form of ephemeral activations which pass fm each node

LSTM introduces an intermediate type of storage via the memory cell

- Memory cell is a composite unit

↳ built from simpler nodes in a specific connective pattern

↳ w/ the novel inclusion of multiplicative nodes

GATED MEMORY CELL

~~equipped~~ memory cells are equipped w/ an internal state & multiplicative gates that determin

① input gate: should given input impact internal state

② forget gate: internal state flushed to 0

(3) output gate: internal state of given neuron should be allowed to impact the cells output

- input
- forget
- output

the key distinction between vanilla RNN & LSTM

↳ support gating of the hidden state

Dedicated mechanisms for when a hidden state should be updated

↳ or be reset

these mechanisms are learned

- e.g. learn to retain important early info

learn to skip irrelevant info

learn to reset latent state

the data feeding into the LSTM gates are the // current input & hidden state of prev time step

the values of the input, forget & output

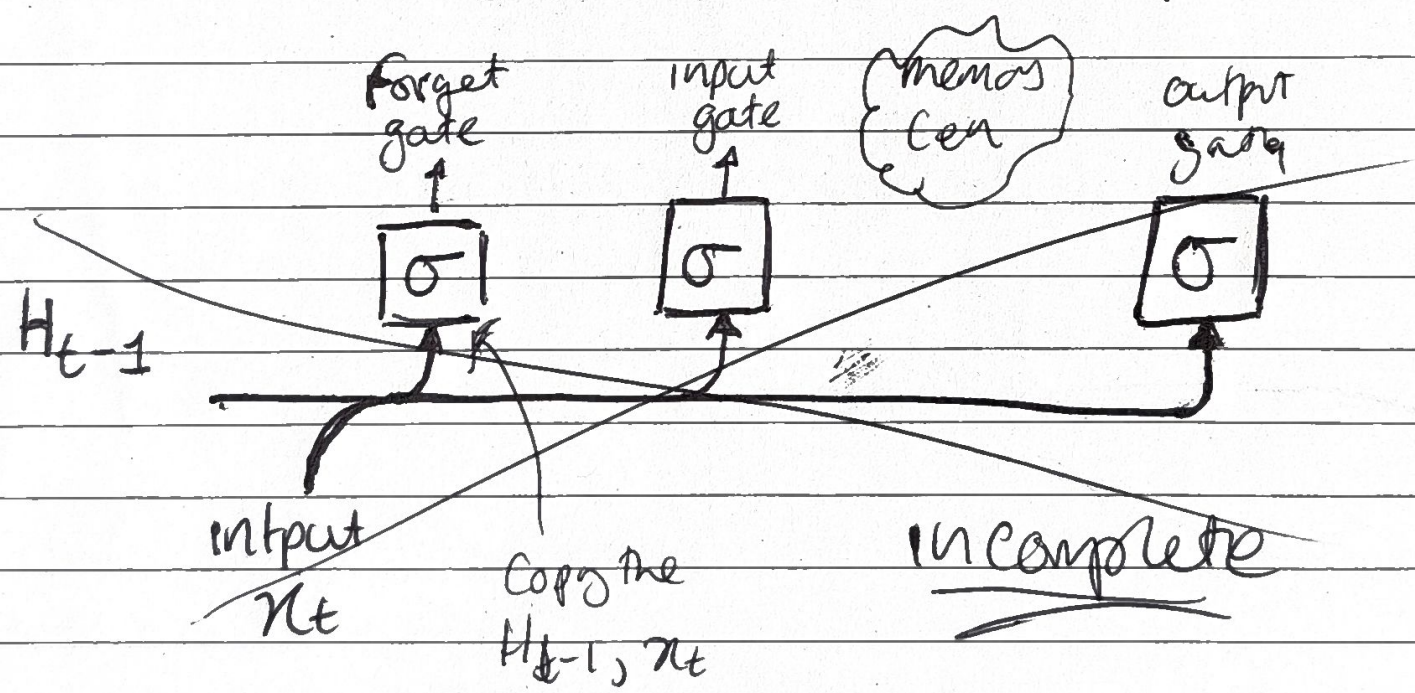
are determined by 3 fully connected layers w/ sigmoid activ

Sigmoid gives the gate a "percentage"

input gate determins how much of input nodes value should be added to the current memory cell state

forget gate = keep current memory or flush it

Output = Whether memory cell should influence the output @ the current time step



$$I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$$

etc.

each gate has its own params $(W + b)$'s

$\tilde{C}t$

Input node

there is also the input node

its computation is similar to that of the 3 gates described above

but uses tahn function

Memory cell internal state

the memory internal state is given by Ct

runs along the top of the cell

input gate governs how much we take into Ct

forget gate decides how much of $Ct-1$ is retained

$$Ct = f_t \odot Ct-1 + I_t \odot \tilde{C}t$$

input Node

if forget=1 & input=0
$Ct-1$ will remain constant

for LSTM the f, I, O params are fixed for all sequences — they are not retrained / vary

Hidden State

need to define how to calc the output of the memory cell // H_t

which is the thing passed onto future cells

① apply tahn to memory cell internal

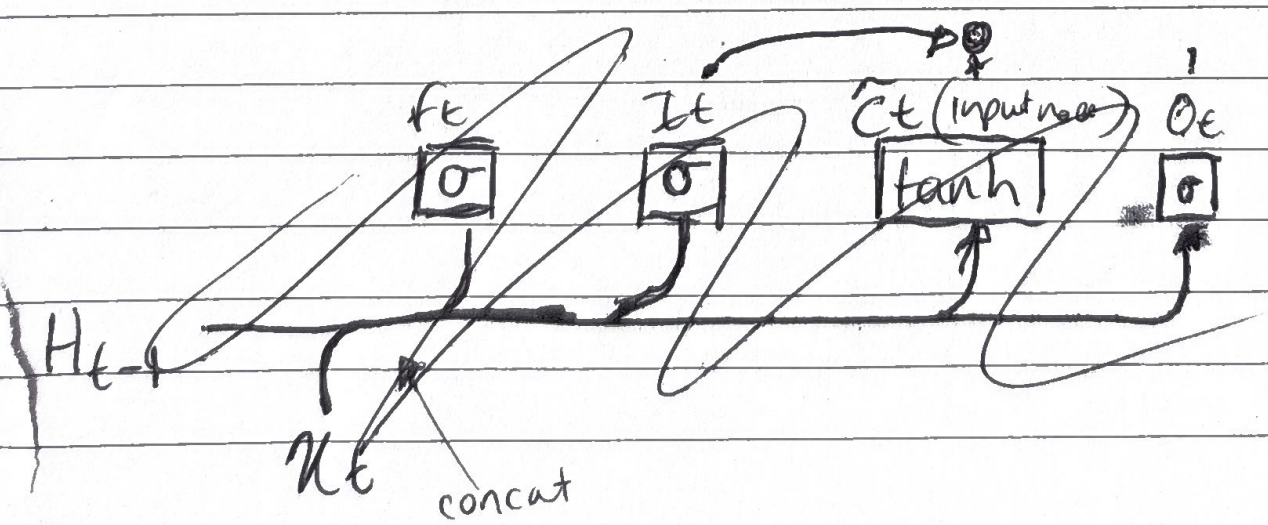
② Point wise multiplication w/ output gate

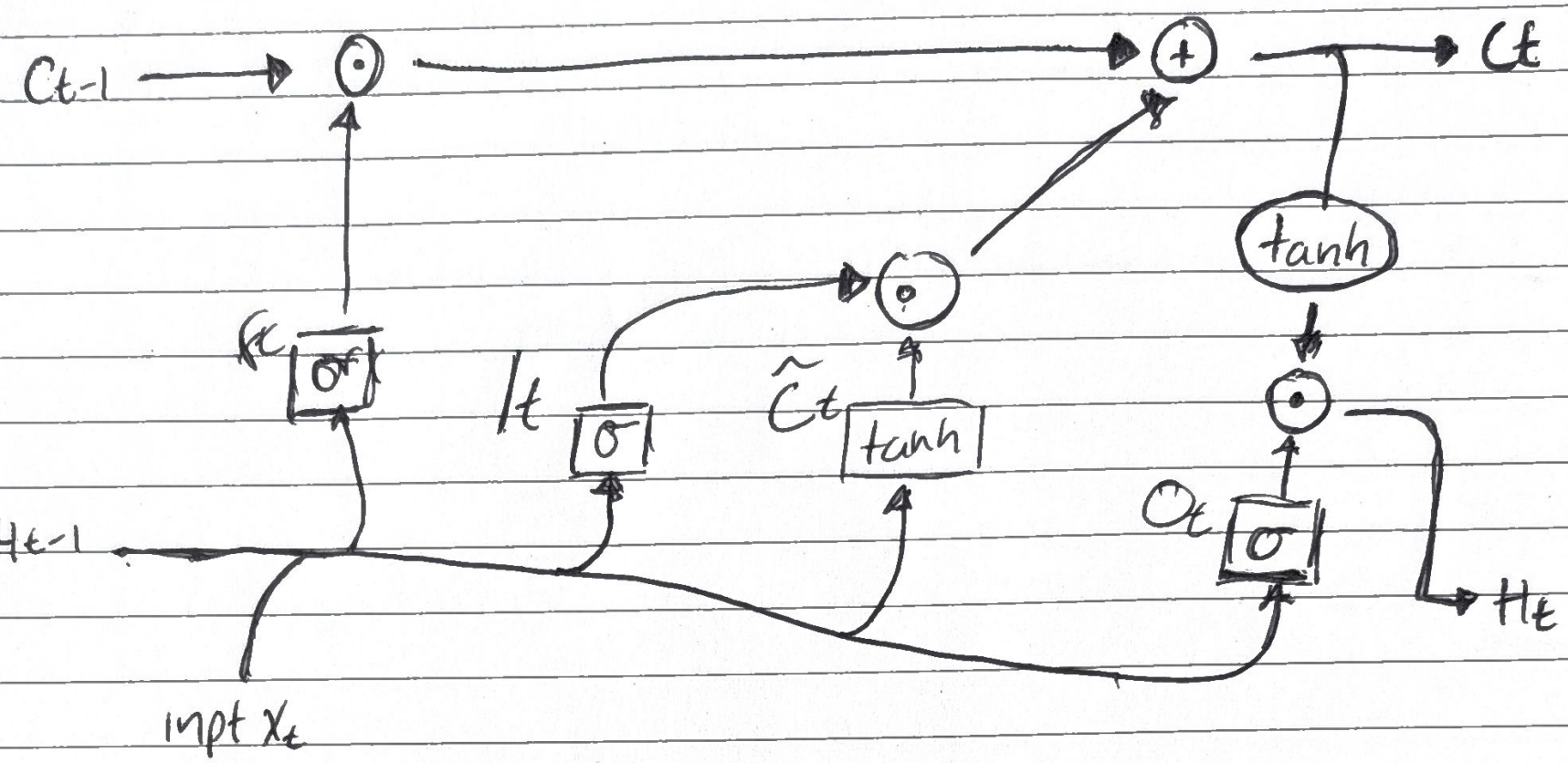
$$H_t = O_t \odot \tanh(C_t)$$

if Output gate O_t close to 1 the the internal state is able to impact subsequent layers

the internal state can accrue for many steps but not impact the H

↳ however is the output gate flips it will suddenly all impact H





10.2 Gated Recurrent Units (GRU)

As RNN & LSTM gained popularity in the 2010, researchers wanted to retain internal state & multiplicative gating but speed up computation

GRU (2014) offered a streamlined version of the LSTM memory cell

Comparable performance but quicker

Reset gate & update gate

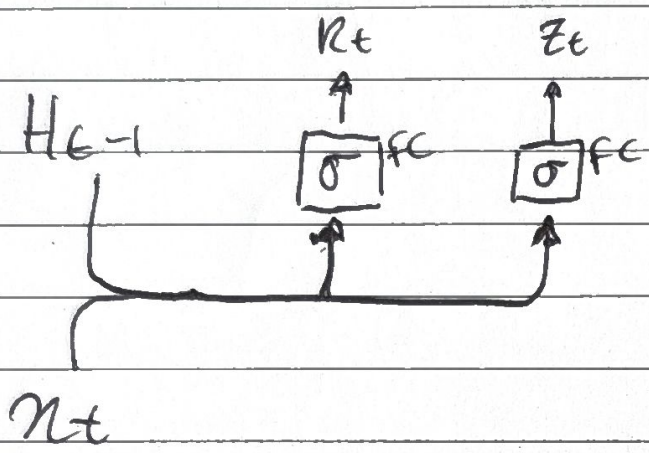
One 3 forget, input & output gates one replaced by 2

Still use sigmoid activations

reset gate decides how much of the previous state to remember

update decides how much of the new state is just a copy of the old one

①



Candidate Hidden State

next integrate the reset gate w/ the regular updating mechanism

↳ to candidate hidden state

$$H_t = \tanh\left(X_t W_{xh} + (R_t \odot H_{t-1}) W_{hh} + b_h\right)$$

the result is a candidate as it still needs an update gate

if $R_t = 1$ = vanilla RNN

Hidden State

Finally need to incorp update gate z_t

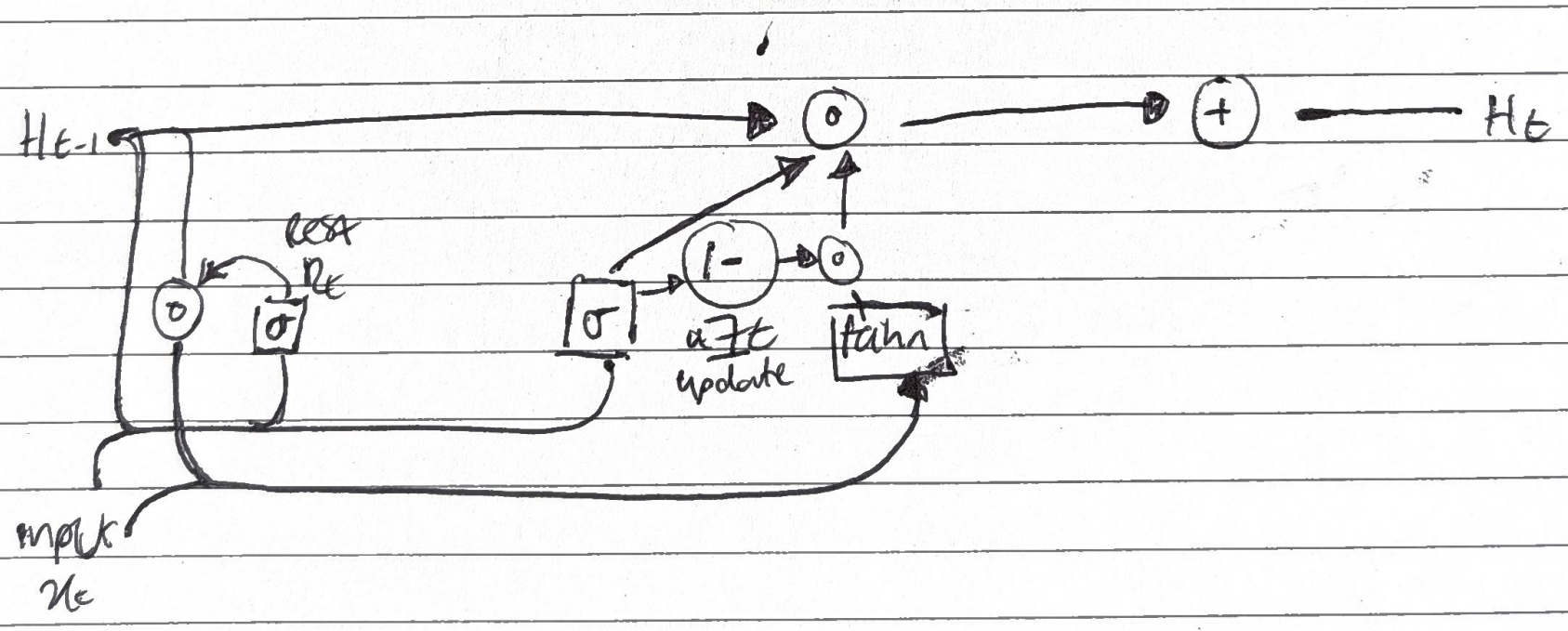
Determins extent to which ~~update gate~~ new Hidden matches the old Hidden

↳ compared to how it matches the new candidate state $\tilde{H}_t$

$$H_t = z_t \odot H_{t-1} + (1 - z_t) \odot \tilde{H}_t$$

So if $z \approx 1$ we retain old state H_{t-1}

if close to 0, approach the candidate latent state $\tilde{H}_t$



① Reset gates help capture short term deps in seqs

② update gates capture long-term deps

10.3 Deep recurrent neural Networks

RNNs inherently seem deep due to the Depth of sequences

However, we wish to retain the ability to express complex relationships between the input & output of a given time step

We want to construct RNNs that are deep time wise but also in the input-output Direction

method is simple stack RNNs on top of each other

each output of a layer is the input to the next

$$H_t^{(l)} = \phi_l\left(H_t^{(t-1)} W_{xh}^{(l)} + H_{t-1}^{(l)} W_{hh}^{(l)} + b_h^{(l)}\right)$$

At the end (top) the output is based on the H_t of the L^{th} layer

$$O_t = H_t^{(L)} W_{hq} + b_q$$

Hidden layer & units are tunable Hyperparas

widths (h) = 64 to 2056

Depth (L) = 1 to 8

10.4 Bidirectional RNNs

Not relevant for task

10.5 Machine translation & the Dataset

tokenization not relevant for task

Mostly an intro to a dataset so no notes

10.6

10.6 encoder - Decoder

w/ seq2seq is the inputs & outputs are of varying lengths then the standard design is encoder- decoder arch

10.7 Seq-2-Seq for machine translation

example where both encoder- Decoder are LSTMS

encoder will take variable length seq as input and transform it into a fixed-shape hidden

During training, decoder is trained against ground truth (Preceding)

i.e. IF RNN predicts wrong token for next seq it will be given the correct token to guess the next from

During test, the decoder retains its own prediction iteratively

10.7 teacher forcing

10.7.2 encoder

transforms an input sequence of var length into fixed shape vector context variable c

the context variable is just the hidden state of the encoders RNN representation after processing the final token of the sequences

10.7.3 Decoder

For each time step, the decoder assigns a predicted prob to each possible token occuring @ $t+1$ step

↳ conditioned upon the ~~prev tw~~ steps prev tokens it in target seq

+ the inputted context variable

(1) ~~Fin~~ final hidden state @ final time step of encoder is used as init of decoder Hidden state

(2) Above requires RNN of E & D to have same number of Hidden layers & units

(3) Context var is concated w/ Decoder Hidden state @ all times

↳ further encodes input seq

↳ Decoder Hidden state starts as C but is updated over steps

(4) to predict prob dist of output token — use A Fully connected layer - (Hidden state transform)

10.7.5. loss function w/ masking

at each time step, decoder predicts a prob distribution for the output tokens

apply softmax to obtain the distribution

calculate the cross entropy loss for optim

10.7.7 Prediction

to pred the output seq at each step

the predicted token from the previous time step is fed into the decoder as an input

simple strat

Sample whichever token that has be assigned by the decoder w/ the highest prob @ each step

in training, @ the initial time step is the <bos> token

<bos> Predict X

X Preds Y

Repeat until <eos>

10.7.8 Evaluation of Predicted Sequences

evaluation trans seq w/ pred seq

but what is appropriate measure?

(2002

Bilingual Evaluation Understudy (BLEU)

- originally proposed for machine trans results

measures qual of outputs of sequences

in principle, for any n-gram in pred, BLEU evals whether n-gram appear in target

10.8 Beam Searches

greedy strat of test time prediction =
@each timestep select the token w/ the highest given probability
- until reach <EOS>

Section = formalize greedy search strat & ident some common probs

two alts: Exhaustive & Beam

@ any time step t'

Decoder outputs predictions representing the prob of each token in the vocab coming next $y_{t'+1}$

conditioned on the prev tokens $y_1, \ldots, y_t$

and the context variable c
↳ produced by the encoder

10.8.1 Greedy Search

$$y_{t'} = \underset{y}{\operatorname{argmax}}\ P(y \mid y_1, \ldots, y_{t-1}, c)$$

this is a reasonable approach & comput undemanding

but it might be reasonable to search for most likely sequence
↳ not token

greedy search does not gaurentee most likely sequence

changing token selected changes the subs of predicted tokns

tokens can be add to get seq prob
most likes seq should be chosen

10.8.2 Exhaustive Search

If we want most likely seq

enumerate all possible output seqs & their conditional probs & select highest

This would be way too expensive

10.8.3 Beam Search

Beam = between efficiency of greedy search & optimality of exhaustive

Bize size hyperparam = k

@ first step select k tokens w/ highest pred probs

k candidate ~~tok~~ first tokes

each sub pred of each route also takes k nexts