Neuron, perceptron, neural net

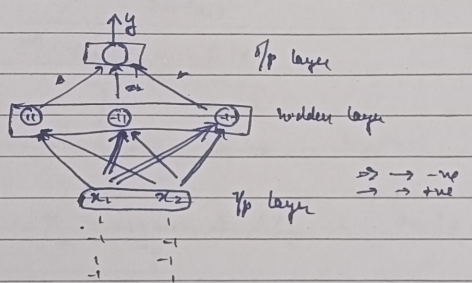
Page No.:
Date: / /

Why —
- Threshold value
- weights
- response is binary
- linear - nonlinear

— Perceptron
— Perceptron learning algorithm → finding optimal weights
→ still has conflicts in implementing XOR (non-linear)

refinement theory

Multi-layer perceptron Architecture (MLP)



$w_0 = -\theta$

$x_0 = 1$

$y = 1$ if $\sum_{i=0}^{n} w_i x_i \geq 0$

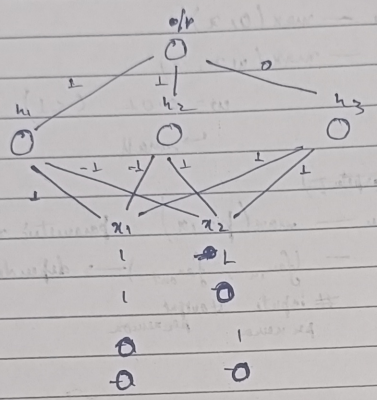
$= 0$ if $\sum_{0}^{n} w_i x_i < 0$

how to determine num layers and num neurons

Activation fn

Page No.:

Date: / /



$x_1 - x_2 = -2$ or 2

$\Rightarrow$

$1 \quad -1$

$-1 \quad 1$

$b = -2$

$h_1 : x_1 - x_2 + b = 0$

$h_2 : x_2 - x_1 + b = 0$

$h_3 : x_1 + x_2 + b = 0$ (2)

$b = -1$

$+h_1 + h_2 + 0 \cdot h_3 + b = 0$

XOR

$x_1 + x_2 = 1 \quad x_1 - x_2 = -1$ or 1

$x_1 + x_2 \neq 2$ or 0

$h_1 : x_1 - x_2 - 1 = 0$

$h_2 : x_2 - x_1 - 1 = 0$

$h_3 : x_1 + x_2 - 1 = 0$

$y : 1 \cdot h_1 + 1 \cdot h_2 + 1 \cdot h_3 - 2 = 0$

Relu $\longrightarrow$ $R \to [0, \infty)$

$$\begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } w \end{cases}$$

no upper bound

Not differentiable at 0

Other loss functions may take more time to reach goal that's why we generally associate x loss function with y problem.

Page No.:
Date: / /

~~num class~~ # class = # o/p layer neurons (3 onwards) [illegible]

relu — max(0, x)

leaky relu — max(wx, x)

w = 0.01 (< 1)

→ small

~~relu (aspiln?)~~

prelu — max(px, x) parametric relu

srelu — (fan_{in}, fan_{out}) → dependent on this

inputs per neuron, # output per neuron

tanh — $\frac{e^{x} - e^{-x}}{}$

- center — 0
- slope will decrease around the center more rapidly
- next-gen sigmoid due to better / clear distinction b/w 0, 1

Updates in Backpropogation

a_i^j → activation fn

h_i^j → $\Sigma a_i x_i$

j → layer no.

i → neuron no.

contribution of each parameter: $\frac{\partial L}{\partial a_i^n} \times \frac{\partial L}{\partial h_i^n} \times \partial$

$$\frac{\partial L}{\partial w_1} = \underbrace{\frac{\partial L}{\partial a_i^3} \times \frac{\partial a_i^3}{\partial h_i^3}}_{\text{o/p layer}} \times \underbrace{\frac{\partial h_i^3}{\partial w_i^2} \times \frac{\partial w_i^2}{\partial a_i^2} \times \frac{\partial a_i^2}{\partial h_i^2}}_{\text{hidden layer}} \times \underbrace{\frac{\partial h_i^2}{\partial w_i^1}}_{\text{i/p layer}}$$

Multilabel Multitask

Loss fn depends on prob.

Page No.:
Date: / /

Hyperparameters

- Grid search

→ Number of Hidden layers, I/p layer, O/p layer
relation b/w i/ps — #i/p dim — ↳ # classes/labels

→ # Neurons / layer

→ Activation fn
→ Vanishing gradient → too small
→ Exploding gradient → too large

→ Weight Initialization
① Xavier Normal: $\sigma^2 = \frac{2}{n_{in} + n_{out}}$
② Xavier Uniform: $b = \sqrt{\frac{6}{n_{in} + n_{out}}}$

→ Xavier init or Glorot init
Glorot → random numbers from known distribution
→ numbers should be in a particular range
→ $\sigma^2 = 1/fan_{avg}$
→ $fan_{avg} = \frac{(fan_{in} + fan_{out})}{2}$
→ Activation - None, tanh, sigmoid, softmax
⇒ ↳ taking these due to symmetric behaviour

init wt = 0
if we init with 0 → no change in gradient → no learning → same o/p → problem

init wt = 1
all i/p treated equally → no change → not just for 1 for any constant → problem

init wt = random
unstable gradients if variance not in check

He → Activation → ReLU and variants
→ $\sigma^2 = 2/fan_{in}$
→ Based on obs that 50% neurons are switched off

LeCun → Activation → SELU
→ $\sigma^2 = 1/fan_{in}$
→ Orthogonal

GBLU

QR Decomp | LLM wt init | Normalization & Standardization

If i/p distribution changes in test and train, we need normalization

Page No.:
Date: / /

Nonlinear wt ( )

Orthogonal → ~~wt~~ random wt matrix W

→ W = QR (QR decomposition, SVD can also be used)

→ ~~QQ^T = I~~ $WW^T = I$ (constraint)

→ resists exploding

→ Unsupervised Pretraining

train layer by layer

			H_3
	H_2		H_2 fr
H_1	H_1 frozen	→	H_1 fr → ...
i/p	i/p		i/p

⇒ Gradient clipping

→ Manage gradients by placing bounds.

if val is in range take it

if beyond threshold, take threshold ← this may change direction of loss fn movement

another approach is to ~~take~~ normalize the weights so that they are within the range

this keeps the direction same as ~~before~~ what it would be if we didn't clip.

⇒ Batch Normalization

→ We want to maintain the same distribution for the inputs of all the hidden layers.

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

μ → mean

σ^2 → variance

γ → o/p scale parameter vector

β → output shift

$$z = \gamma \otimes \hat{x} + \beta$$

z → o/p of ~~from~~ BN algo

page rank freq
$R \propto \left(\frac{1}{f}\right)^{\beta}$
$\log R = -\beta \log f$

ML: Intro to Nonconvex Optimization | Rishabh Bhardwaj

Norm/Rat — Norm/range 0-1 — μ, σ^2
BN vs LN vs Norm vs Std

Page No.:
Date: / /

Batch v/s Layer Normalization || Normalization v/s Standardization

Gibbs, ibbs, Power law

→ Normalization can be done on the basis of # neuron and ~~layer by layer~~
also layer normalization
↳ # layer

BN handles both expl and vans. problem
LN handles only one (check)

Optimizer

→ Improvement on GD

Scheduler → manages/adjusts LR

~~Power, exp~~ c = power

Power Scheduler: $\eta(t) = \eta_0 / (1 + t/s)^c$, t = iteration number, s = total iters

Exponential scheduling: $\eta(t) = \eta_0 \cdot (0.1)^{t/s}$

Piecewise constant scheduling: $\eta_0 = a_i$ for k epochs, then a smaller LR for another number of epochs

Performance scheduling: measure val err. every N steps and reduce LR by a factor λ (can be loss fn or ability fn)

SGD with Momentum

initially: $w_{t+1} = w_t - \eta \Delta w_t$

with momentum: $w_{t+1} = w_t - v_t$ → $v_t = \beta \times v_{t-1} + \eta \Delta w_t$

$\beta \in (0, 1)$

→ tackles — local minima
— saddle point
— High curvature

~~$\beta(\eta \Delta w_{t-1}) + \eta \Delta w_t + \cdots$~~

~~β^{t-1}~~ ~~$\beta(\beta v_{t-2} + \eta \Delta w_{t-1}) + \eta \Delta w_t$~~

~~$\beta^t \eta$~~

~~$= \eta(\Delta w_t + \Delta w_{t-1}\beta + \cdots$~~

~~$= \eta \sum_0^t \beta^i \Delta w_{t-i}$~~

~~$+ \sum \beta^i v_{t-1}$~~

Page No.:
Date: / /

⇒ Nastrov Accelerate Gradient

→ Look ahead

$$w_{la} = w_t - \beta v_{t-1}$$

$$v_t = \beta v_{t-1} + \eta \Delta w_{la} = (w_t - w_{la}) + \eta \Delta w_{la}$$

$$w_{t+1} = w_t - v_t$$

→ May get stuck in local minima (check why?)

⇒ AdaGrad (verify?)

$$w_{t+1} = w_t - \frac{\eta \Delta w_t}{\sqrt{v_t + \epsilon}} \rightarrow (?) \text{ check}$$

$$v_t = v_{t-1} + (\nabla w_t)^2$$

→ Not preferred in Deep NN

⇒ RMSProp (check)

similar to Adagrad → just add decay

$$v_t = \beta v_{t-1} + (1-\beta)(\nabla w_t)^2$$

⇒ Adam (check)

$$w_{t+1} = w_t - \frac{\eta \times m_t}{\sqrt{v_t + \epsilon}}$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) \nabla w_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla w_t)^2$$

CNN from Hands On

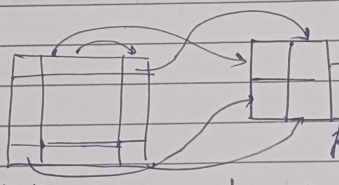
Page No.:
Date:

Regularization

Dropout → n% neurons are switched off / disabled
→ don't emit op or gradient
→ sparsify the net
→ explore multi. architectures using dropout → ensemble

CNN

→ If we process images ~~as~~ in the regular way, we need such nodes ~~for each~~ i.e. 1 neuron for each pixel



Kernel has same weights throughout channel, each kernel for one have various wts.
Padding → To reduce bias of common area

→ dimensionality reduction images

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in} = # i/p features
n_{out} = # o/p features
k = conv. kernel size
p = conv. padding size
s = conv. stride

→ 3 Filters Vertical Horizontal etc.

Page No.:
Date: / /

→ We can use n ~~kernels for n~~ different kernels and get n diff. feature maps to capture more features

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{w'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \quad \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

16×16 i/p 3 channels m×m filter x filters
each filter gives m×m → 3D x · m×m

→ Back propagation

~~Pooling~~
Pooling → Mathematical aggregation fn
→ No learning
→ kernel is manual throughout
→ No idea about loss and ~~value~~ info. of each pixel inside my kernel
→ Adv. → dim. red.

Translation invariance → Pattern / feature learnt can be identified at any location
→ Good for object detection

ARCHITECHTURES
- AlexNet
- ResNet
- LeNet
- ... All till GoogleNet

Ch14 HOML

Page No.:
Date: / /

ANN
CNN
RNN
LSTM
GRU

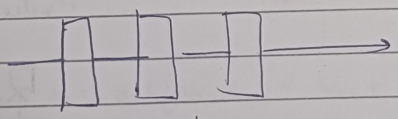
Data Augmentation → flip, shift, shrink, brighten etc.

Local Response Normalization (in AlexNet)
(Formula from book)

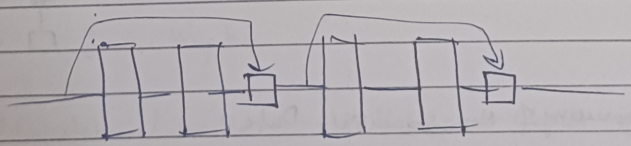
Where else can we use CNN
→ Tabular data
→ 1D data

→ But other techniques give better results for Tabular data.

Mark / Forward mark



DNN



Residual

Xception

No. of Parameters numericals: 4F 4

Each conv: Params = $(K_w \times K_h \times C_{in} + 1) \times C_{out}$

Fully connected: Params = (input units × o/p units) + o/p units

If BN: Params = 2 × num features

~~$(K_w \times K_h \times C_{in} + 1) \times C_{out}$~~

Ph-3 — 14-20 Nov — Lab - one on one eval — 404B EDL — 15-30 mins
Slot will be given
Report + Repo + Readme + code + what we did
(Template)
(20% plag)

Page No.:
Date: / /

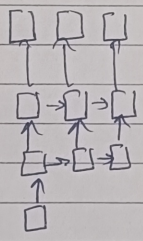
RNN Recurrent Neural Network

→ Regular fed-forward networks can not capture sequential relations.

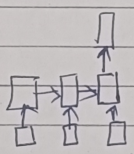
→ Theoretically, CNNs can work, but it is not quite right.

Why CNN
→ for spatial relations
→ for same input
→ for getting proper response for non-target containing inputs

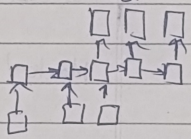
One to many (image captioning)



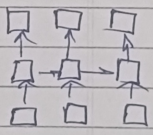
many to one (video action prediction)



many to many (video captioning)

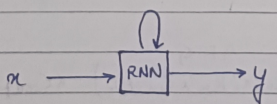


video classification



Sequential Processing of Non-Sequential Data
↳ RNN can classify MNIST dataset (given as one image)

RNN



RNNs have an internal state that is updated as a seq. goes on

Compare different outputs from different activation functions

Page No.:
Date: / /

Vanilla RNN

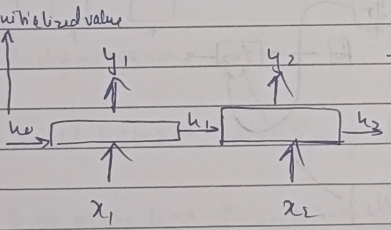
Hidden state update

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

h_t ← new state; f_W ← neuron / some fn with parameters W; h_{t-1} ← old state; x_t → input vector at some time step

$y_t = f_{W_{hy}}(h_t)$ (like activation (f))

y_t ← output; $f_{W_{hy}}$ ← another fn with parameters W_{hy}; h_t → new state



Elman RNN or Vanilla RNN

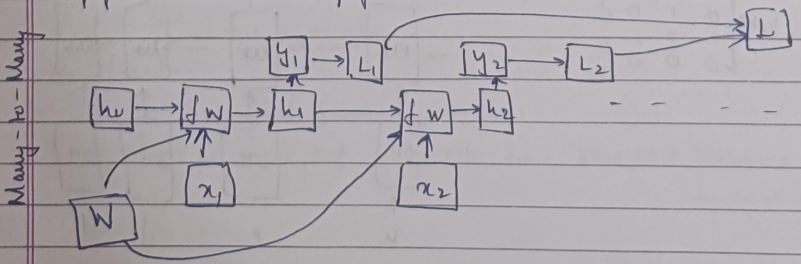
Eg: $h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$

$y_t = W_{hy} h_t$

Q. Why tanh
Why not ReLU
GeLU
Sigmoid

Computational Graph → Static → Complete required
→ Dynamic → Layer by layer - output

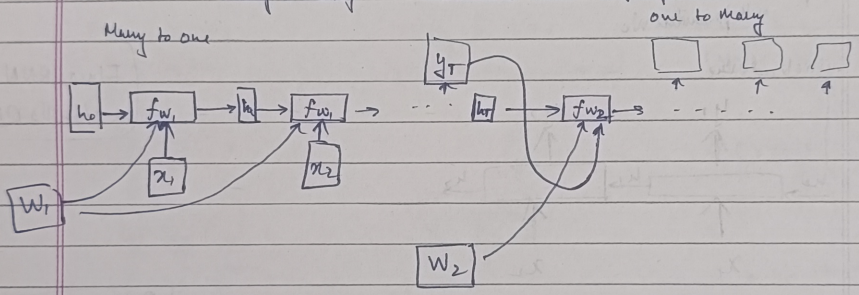
Map of operations that will be performed



Similarly Many to One (2 types)
One to Many

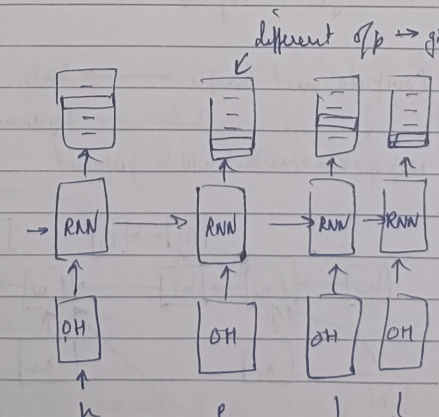
In one-to-many, what would be the input for the intermediate / non-initial nodes → replicate input (based on problem / modality)
OR → replicate output ☑
✗ → Garbage value (not preferred → garbage in garbage out)
→ 0s or 1s (☑ Since W is same for all, it would not be ~~beneficial~~ beneficial)

Sequence to Sequence → Many to One + One to Many
Many to Many → character level language model



Char - Lvl — LM

Train

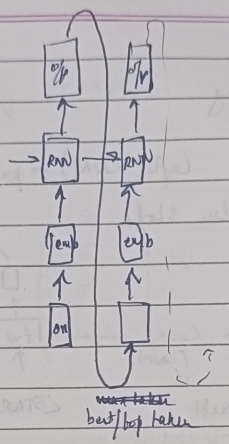


Hello → [h, e, l, o]

one hot →
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

Page No.:
Date: / /

Text



Backpropagation

In std. backprop → contribution of first neuron reduces drastically → vanishing

Better approach → Truncate → calculate in chunks

RNN Tradeoffs

Advantages

- ↳ can process any length i/p
- ↳ computation for step t can use info from prev. steps
- ↳ Model size doesn't increase for longer i/p
- ↳ same W on each

Disadv.

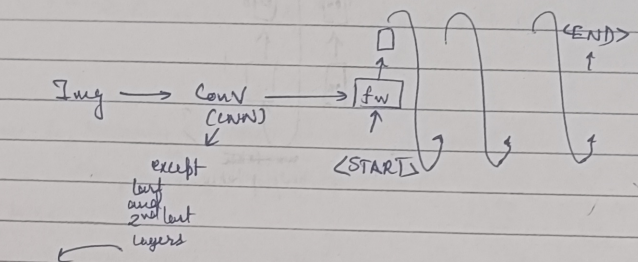
- ↳ Sequential processing → slow
- ↳ previous info & best ago some time ⇒ + backprop is diff due to vanishing grad.

15-18 words can be worked with nicely in vanilla RNN

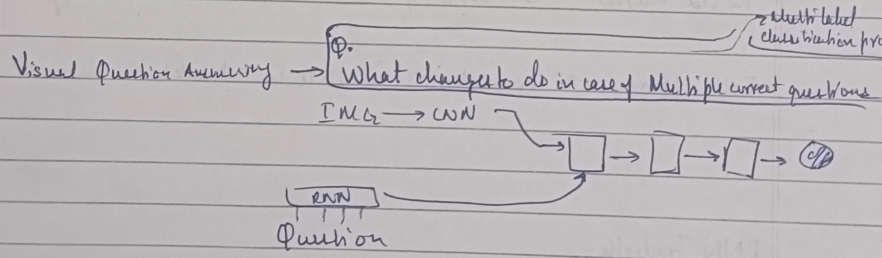
Page No.:
Date: / /

Image Captioning

→ Image as input (after CNN) → problem: same i/p for all
→ Images as Hidden State

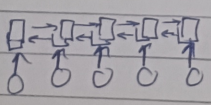


Why?
→ They are task specific → ~~have~~ needed for classification



Vis QA → Dataset Bias
→ Generalization

Multilayer RNNs ~~(VERIFY)~~ (VERIFY)



GC

E.S. Code may also be asked for End-Sem exam

END-SEM

Page No.:
Date: / /

Long Short Term Memory (LSTM)

g → Gate gate → how much info to write to cells

c → cell state → long term

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

i → input
f → forget
o → output
g → vanilla rnn

update ← forget
in prev info
o/p ← output

$$c_t = f \odot c_{t-1} + i \odot g$$

(f → delete prev stored; i → place this instead; g → for determining how much to write)

$$h_t = o \odot \tanh(c_t)$$

Vanilla RNN Gradient Flow

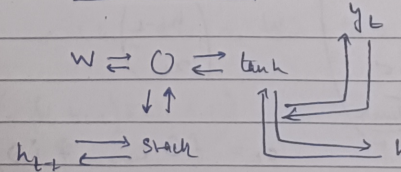
Backpropogation Through Time → calc. and backprop at each step (no need to wait for end)

$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{xh} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'\left(W_{hh} h_{t-1} + W_{xh} x_t\right) W_{hh}$$



$$\frac{\partial h_t}{\partial W} = \tanh'\left(W\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)\begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T}$$

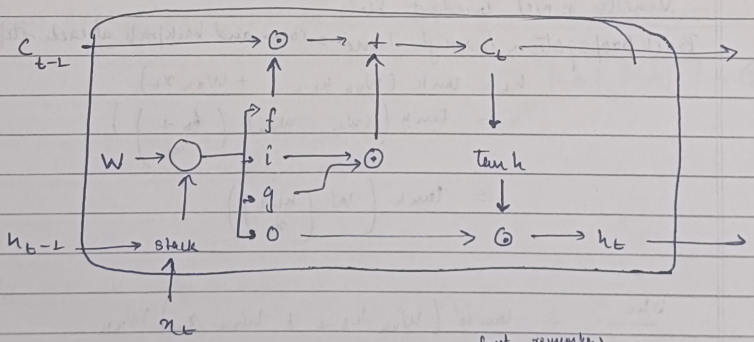
→ have to change RNN structure

Vanishing gradients for tanh sigmoid

Exploding for ReLU leaky ReLU

↳ Gradient Clipping

⇒ LSTM



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

→ vem ou forget (0 forget, 1 remember)

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

↳ why → capture more → help / del

LSTM — 43 words

Page No.:
Date: / /

C_t backprop has no W multiplication, so easy to calculate

Do LSTM solve the gradient problems
→ LSTM makes RNN (training for) to retain/process info

→ LSTM doesn't guarantee freedom from vanishing/exploding gradients.

→ LSTM Gradient Flow.
Backpropagation in LSTMs (VERIFY)

$$h_t = o \odot \tanh(f \odot c_{t-1} + i \odot g)$$

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} \begin{bmatrix} W_1 & W_2 \\ \sigma W_1 & \sigma W_2 \\ \vdots & \vdots \\ \tanh W_1 & \tanh W_2 \end{bmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$L \rightarrow \frac{\partial L}{\partial y} \frac{\partial y}{\partial}$

$$\begin{aligned} &\sigma W_1 h_{t-1} + \sigma W_2 x_t && i \\ &\sigma W_1 h_{t-1} + \sigma W_2 x_t && f \\ &\sigma W_1 h_{t-1} + \sigma W_2 x_t && o \\ &\tanh W_1 h_{t-1} + \tanh W_2 x_t && g \end{aligned}$$

$$h_t = \sigma W_1 h_{t-1} + \sigma W_2 x_t \odot \tanh\left(\left(\sigma W_1 h_{t-1} + \sigma W_2 x_t\right) \odot c_{t-1}\right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = \left(\sigma'(W_1 h_{t-1}) W_1 + 0\right) \odot \tanh'(\cdot)\left(\sigma W_1 c_{t-1} + \sigma W_1 c_{t-1}\right) + \sigma'(W_1 h_{t-1}) W_1 + \tanh'(W_1 h_{t-1}) W_1$$

$$+ \sigma(\cdot) \tanh'(\cdot) W_1 + \sigma'(\cdot) \tanh(\cdot) W_1$$

$\Rightarrow \sigma'(\cdot) W_1$

55/100 = 2.75/5   2.78 → 22   70 → 3.5/5   34.75/40   ~~48.75/45~~

51/3
17 34.75/20
15+2.75 17.75/20

Page No.:
Date: / /

37.25/45
5.75

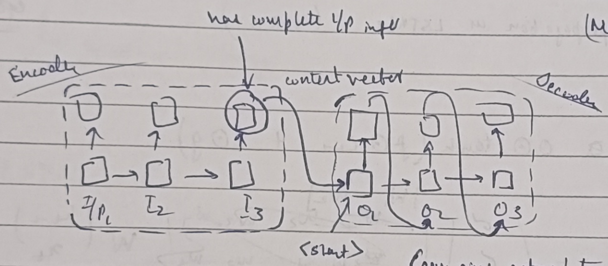
Other RNN variants

GRU → reset gate and forget gate only
r z
└→ 32/38 words

⟹

⟹ Encoder - Decoder Architecture

Generative v Discriminative
(Markov, Bayes) → which is which



(can give actual translation in training phase as I/P to decoder)

Why not put context vec. as hidden state to all?
↳ because prev. node ~~would~~ captures all prev info, CV becomes un-updated if we keep going with that.

- For long sentences context vec alone are not enough, so concatenate all o/p or hidden states (what?) and consider it as context vector.

Problem

~~could~~ concatenation may lead to domination of some inputs. <why?>

Luang Attention

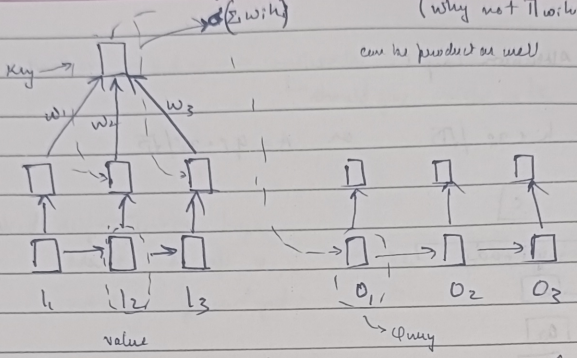
(before self-att)

Page No.:
Date: / /

to handle exploding gradients
to introduce non-linearity

$(\sum_i w_i h_i)$ (why not $\prod w_i h_i$)

can be product as well



query key value

(check)

(= compare with attention)

dictionary analogy

$d = \{a: 1, b: 2, c: 3\}$

key — value

$d[a]$ — query

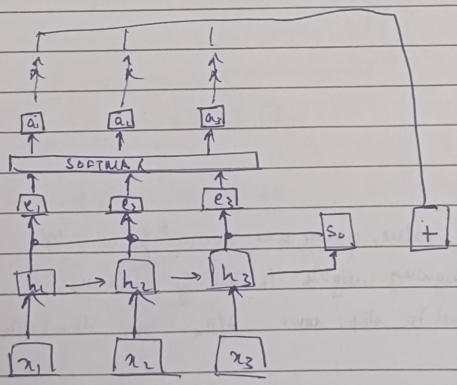
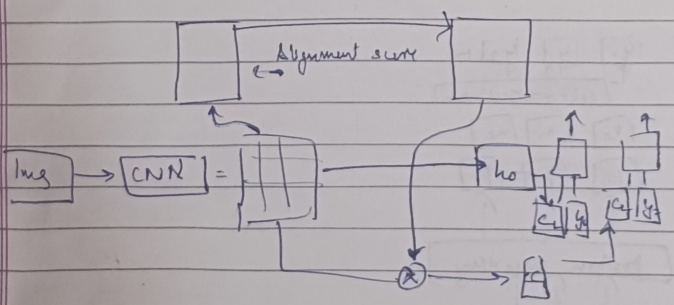


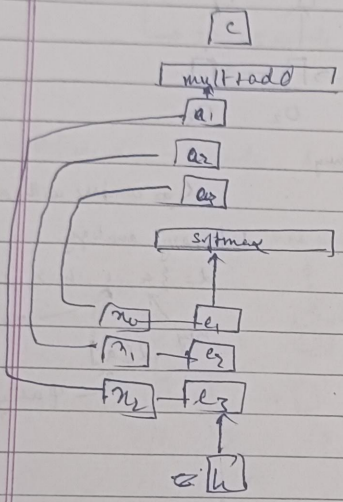
Image Captioning using Spatial features



Page No.:
Date: / /

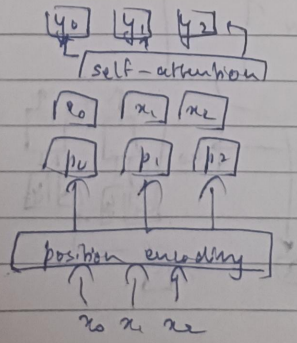
Causal Attention Layer

$A = k \cdot x_i / \sqrt{D}$ or $A = q_i \cdot x_i / \sqrt{D}$



self Attention → key, value, query are derived from i/p
→ Assigning weights to identify relations and to stop some i/p's from dominating

Position Encoding



Relation b/w residual connections and layer normalization

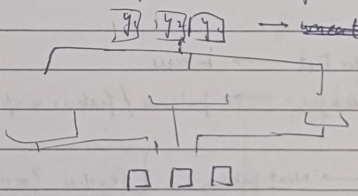
Residual vs RNN

Page No.:
Date: / /

position encoding → constants
↳ should give unique vals

Masked self-attention layer
Mask & all x_{i+k} $\forall k \geq 1$ because those i/p have not been found yet.

Multi-head self attention layer



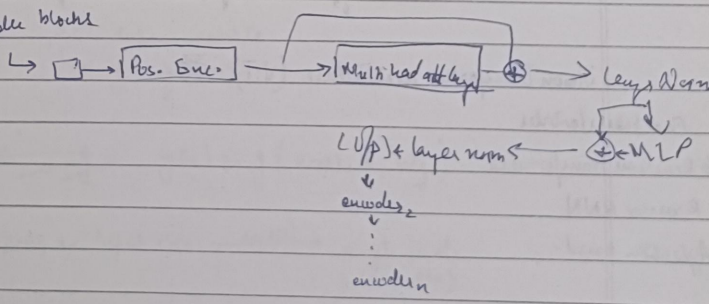
General att v self Att

RNN v Transformer

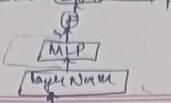
Normalization ensures residuals are added

Transformer

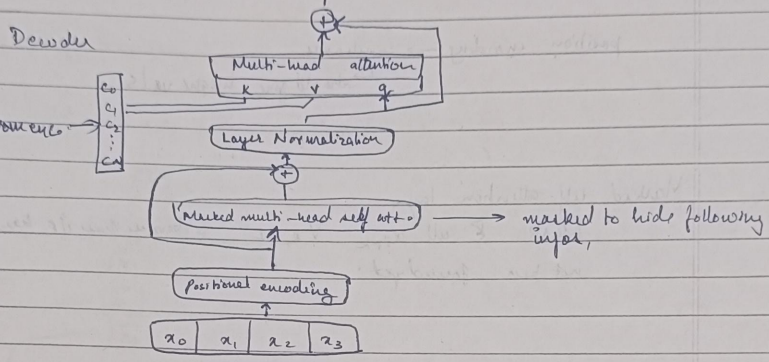
N - encoder blocks



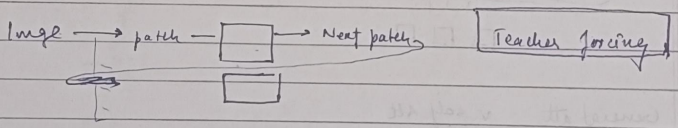
Why we projection in self attention → FC → O/P



Page No.:
Date: / /



tokenizer enforces sequential flow for text → tokens
convolution does same for images → patches / feature maps.



Img → enc → C → has info of entire space

How to handle different kinds of inputs (images, audio, text)

What o/p do we get at each step (what o/p and what type)

Vision Transformer architecture (ViT)

From papers/articles

Recursive Transformers

Recursive RNN

Application based.

Hierarchical

Page No.:
Date: / /

Complexity

Recursive NN → process structured data → can work with topologies (like graphs) rather than feature based approach

initially only worked with acyclic etc. under assumptions structured domain has little work str.

$f \rightarrow$ state transition $\rightarrow$ MLP

$g \rightarrow$ output fn $\rightarrow$ MLP

$$a(v) = f(a(\text{child}[v]), I(v), v, W_f)$$

- $a(\text{child}[v])$ → state transition nn
- $I(v)$ → input label
- v → node
- W_f → Weights model param

$$a(\text{ch}[v]) = \left[a(\text{ch}_1[v]), a(\text{ch}_2[v]), \ldots a(\text{ch}_0[v]) \right]$$

$a(nil) = a_0 \rightarrow$ frontier state

$S \rightarrow$ super source

↳ $y = g(a(s))$

~~$y = g$~~ $= y(v) = g(a(v), v, W_g)$

mapping of input str → RecNN → single node (nn str.)

max transfer → an intelligent sys builds know by max transfer of prev know

max recov. → an int. sys must be able to infer causes

Neated LSTM paper

Page No.:
Date: / /

Recursion in existing architectures

hierarchy → correlate processed info at levels.

info flow + logical rel =

→ sequential prop

E Improve state transition f and g

current pts W ——————→ updates viewed as noise around W

E↑ as var is ~~decrease~~ ↑ of dist. recs

→ goal penalize such wts