Exam prep 2020

MLP

1) Supervised v/s unsupervised

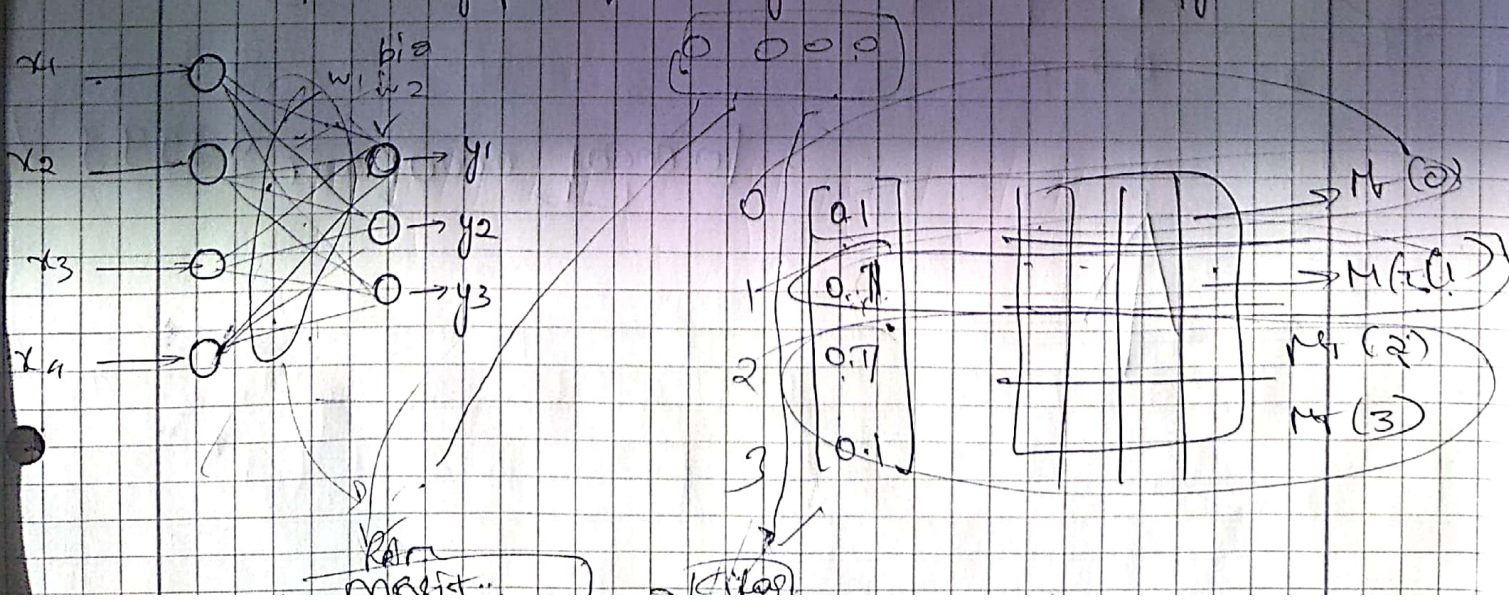
ML Algo	Task	Experience	Performance
Supervised	Approximate a Target function from Training data	Labels in Training data	Confusion Matrix, Accuracy, precision, recall
Unsupervised	Find hidden patterns	No experience during Training. In self supervised it may learn data distribution	Optimization criteria like distance measure

2) Underfitting v/s overfitting

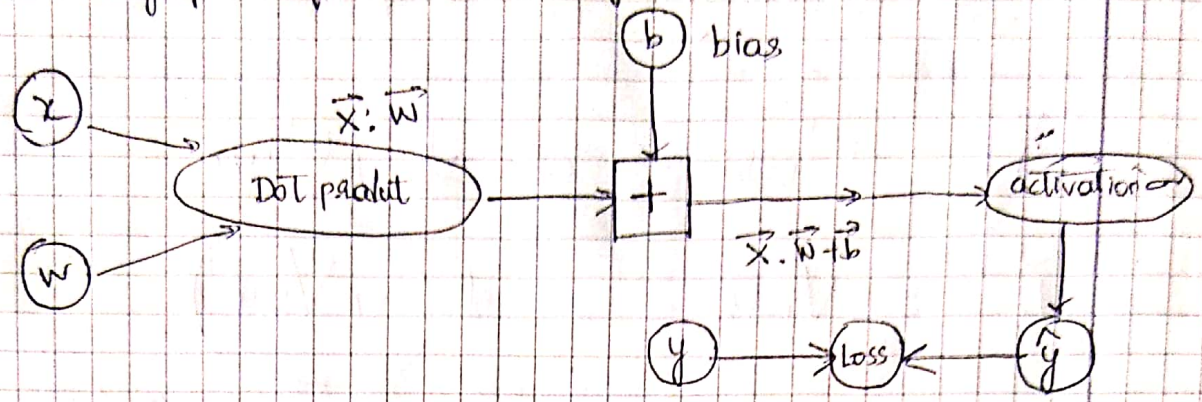
Underfitting: Occurs when you try to overpower your algorithm with your own prior knowledge. Hence causes algorithm not to learn anything from data but only optimize for your knowledge

Overfitting: Occurs when model tries to learn the data too well without caring about noise that could exist in data
Unnecessary variance

3) Sketch computational graph for 1 layer MLP with crossentropy loss



3) Sketch computation graph for one layer neural net and crossentropy



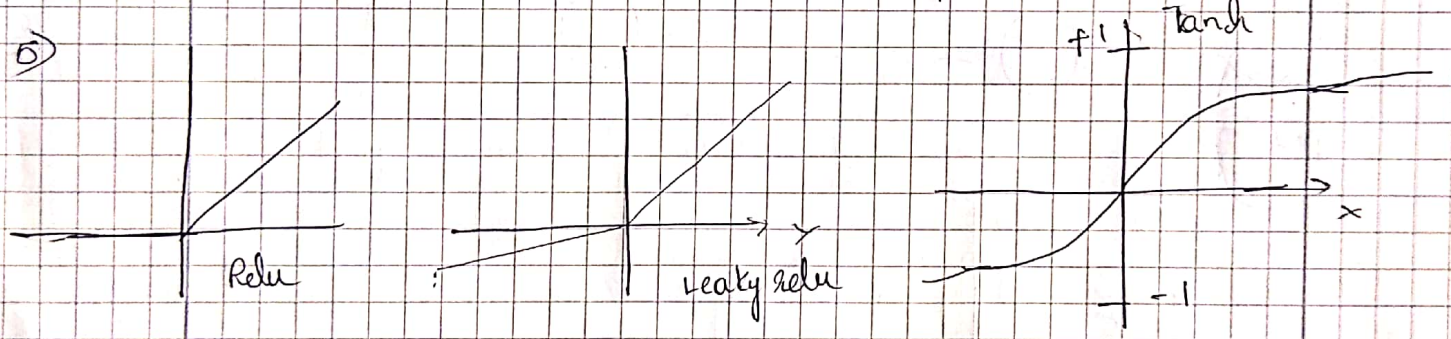
$$Loss = -\sum_{i} p_i \log \hat{p}_i$$

4) Why do we need non-linearities

→ Linear activations are compressible, multiple linear layers can be compressed to a single layer making it not possible to learn complex functions

Nonlinear activations can help learn complex function

5)



Sigmoid :- Works well where we need probabilities as output, But causes saturation

Tanh :- Works a bit better than Sigmoid as it is symetric around zero, Can act linearly if for small activations near zero, saturation still exists for large values

ReLU :- Sparse activations, no saturation in +ve region, Cannot learn if neuron falls below zero

Universal approximation Theorem

My feed forward n/w with atleast one hidden layer and a squashing function can approximate any continous function in subset of R^n

→ But it doesn't talk about how many neurons we need in hidden layer

6) Exploding gradient : When the gradient to be updated is too huge such that once updated we may skip current minima and land on some other minima . The update would be too huge and we would be unable to learn

Vanishing gradient : The gradients to be updated would be so small that we would need lot of steps to learn anything substantial

Remedy : Choice of activation fnc , network architecture

7) Batch v/s stochastic

Batch & Type	Batch	Stochastic
Gradient update	After entire batch	For every example
Gradient value	Avg loss of entire batch	Loss of individual example
Memory	Need lot of memory as need to iterate over entire batch	Need less memory as updating for one example
Number of backprops ~~Time for training~~	Much faster as less number of backprops required	Backprop for every example would slow down training
Training	Gradual , slow and smoother movement towards minima due to averaging of errors	Fast paced and spiky movement towards minima as single example can have lots of noise
Speed of Training	Takes more time to train due to less no of backprops	Takes less time to train but chances are that they can skip local minima due to noise

Remedy : Use minibatch gradient descent to increase batch size & frequency of back prop

8) Backpropagation algorithm is a chain rule based gradient compute method used for efficient computation of gradient/error in DAG graph like systems.

9) Negative log likelyhood is a popular loss function for sigmoid / softmax output as they gel well together. The log in log-likelyhood helps immensely to deal with underflow / overflow issues of small probabilities

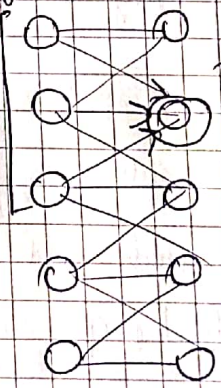
10) Zero initialization with MLP

→ if we use zero initialization then each neuron is gonna produce same output and during gradient update each neuron would be updated in same way, hence neurons cannot learn anything new that distinguishes them from one another

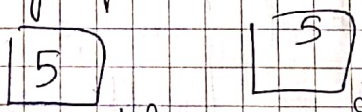
Convolution Neural networks

1) Properties of CNN

i) Local connectivity :- Each neuron in next layer is only connected to a subset of neurons in previous layer



ii) Translation equivariance :- It has ability to detect images and change by equal amount as shift appeared in input. But rotation / shearing cannot be having



doesn't share same principle

iii) Parameter sharing :- The learnt parameters are applied to different parts of image. Hence shared by more than one function of model

2) Why does encouraging activation sparsity helps learn complex filters?

Activation sparsity refers to definite and nice discriminative patterns in feature map. To get this discer distinctive featuremap, our filters too must detect unique patterns that other filters are not detecting Hence this encourages complex filters

3) ReLu is mostly used in CNN why?

ReLU has sparse activation property as any activation beyond zero is made zero helps in getting more distinct feature maps. Also ReLU has only +ve outputs which perfectly syncs with image pixel intensity as intensity is almost always +ve

4) Purpose of 1x1 convolution

→ Reduce number of channels of input whilst preserving its spatial dimensions

6) Padding, Striding, Dialation → Advantages / disadvantages

i) Padding
- → Valid :- Dimension of output reduces to that of i/p
- → Same :- Output featuremap dimension is same as i/p. Some prominance is given to edge pixel
- → Full :- Output map could be larger than i/p, gives equal attention to all parts of grid. Could be used in audio signal analysis

ii) Striding → controls how many times you apply convolution and the positions where you apply convolutions on input. Usually used to downsample

iii) Dialation → Tries to increase receptive field of the feature map w/o increasing the number of parameters (filtersize) by much. Ex:- audio analysis

7) Pooling :- Layer that produces a sort of summary statistics of Input image (avg/max/etc) at local neighbourhood level

→ Pooling in itself doesn't cause significant downsampling, but strides combined with pooling causes

Max pool :- Outputs max element in a local neighbourhood of input. Usually can detect edges / corners / objects much better. Translation invariance to slight changes. Output loses some info about input

Avg pool :- Outputs are averaged local neighbourhoods, hence input info is not lost much. But relatively poorer than maxpool in detecting edges as info is averaged out

8) When to add feature maps & concatenate feature maps

Adding: Normally featuremaps are not added directly if we want to learn a final averaged single featuremap, else they are added in a residual setting where layers in a deep network have to learn when to whether to apply identity function or learn new knowledge & add it. pass info as is

Concatenate: Used in settings where there is a need to reuse featuremaps from earlier layers to add more context like location etc

Recurrent Neural Networks

1) Recurrent Neural Nets, What are they and how are they different from 1-d CNN

Recurrent neural net ~~are~~ as are neural networks that can theoretically store history of arbitrary length and work on data of arbitrary length having some order

→ They differ from 1-d CNN as in theory their receptive field is unlimited and can store memory. Its next output is a function of its previous state which is not the case with 1-d CNN

2) ~~Advantages of~~ RNN

→ Process sequential data which could have some order
→ Capture history of input / states
→ share parameters through a very deep computational graph
  i) Use same update rule at every timestep
  ii) Current o/p is a function of previous o/p

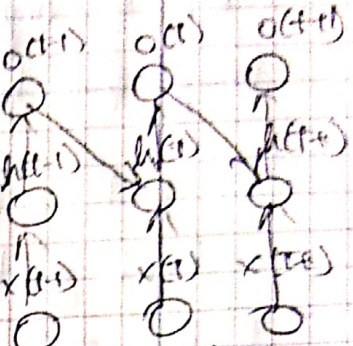
3) Unrolled v/s Rolled network

Rolled n/w: Even though training can be parallelized we may need to learn separate n/w or rules for each combination of sequence
→ Different parameters are used at for different time, hence cannot apply same fnc again & again

→ Current o/p

Unrolled n/w :- same function applied at each time step and parameters are shared. But gradients need to be propagated from output through different intermediate memory states at time steps.

→ Training is always done in sequence



4) Teacher forcing

→ Training method used in training RNNs

→ Previous output is used to calculate current memory

→ Cannot remember anything about past except previous o/p.

⇒ Usually used to store at start of training to make n/w learn the distribution of data (know domain) which helps later when we switch off teacher forcing.

→ Training could be done in parallel

⇒ Don't use it for complete training unless output is extremely rich and high dimensional so that there is less need to remember history.

5) Problems with RNN

→ Training is done step by step in sequence, hence slower & cannot fully use GPU

→ Due to Backpropagation that has to go through all states, usually gradients die out or grow out of control, using same shared parameters adds fuel to this.

$p^{(t)} = \text{Concat}(x^{(t)}, h^{(t-1)})$

$h^{(t)} = \tanh(c^{(t)} * Og^{(t)} +$

$Fg^{(t)} = \sigma($

Forget gate $Fg^{(t)} = \sigma(W_F P^{(t)} + b_F)$

Input/Update gate $Ug^{(t)} = \sigma(W_I P^{(t)} + b_I)$

Output gate $Og^{(t)} = \sigma(W_O P^{(t)} + b_O)$

$\tilde{c}^{(t)} = \tanh(W_c P^{(t)} + b_c)$

$c^{(t)} = Fg^{(t)} * c^{(t-1)} + Ug^{(t)} * \tilde{c}^{(t)}$

6) LSTM and its purpose

→ LSTM is a variant of RNN which tries to solve the gradient problem present in vanilla RNN

→ It addresses these gradient chat challenges by creating a separate path devoid of any major obstacles for activations during forward prop and gradient during backward prop. To propagate This is usually called Constant Error Carousel (CEC) and

→ It protects this path from noise/unwanted input during forward prop irrelevant gradient update for the cell during backward prop using Input and Output gate. They act as switches to only allow relevant info to go forward and relevant info to update cell gradient

Roles and Responsibilities of gates

Input gate: Protects CEC from irrelevant activations/noise that are not necessary for our task

Ex: ~~I visited Nepal in 2009 by Airplane~~
Black pearl is a beautiful ship and Jack Sparrow is her Captain

Task: Gender of proper noun

Gender Blackpearl = ?
Gender Ship = ?
Gender Jack = ?
Gender Captain = ?

→ For the activations of tokens "is" "a" "and" "is" are irrelevant for tracking gender, hence adding them or using them, is equivalent to adding noise. My input gate can filter out such nonsense & only allow required activations.

Output gate: Protects cell from unnecessary gradient updates and also passes down the output to next cell.

Example: As in my previous example: I need not update LSTM when tokens "is" "a" "and" etc occurs as these are articles and irrelevant to gender. I can only let through gradients for personal nouns. This saves

→ This saves unnecessary parameter updates, refining the parameters better

Forget gate :- Initial variant of LSTM did not have forget gate, it was added later to help solve catastrophic activation explosion & saturation problem.

if we don't have forget gate then, theoretically we can remember entire history

$$C^{(t)} = \underbrace{1.0}_{\text{no forget gate} \Rightarrow \text{Remember everything}} * C^{(t-1)} + \text{Ingate} * \tilde{C}^{(t)}$$

$$O^{(t)} = \tanh(C^{(t)}) * \text{Out gate}$$

Problems :- As we go on-and on $C^{(t)}$ gets bigger and bigger since we are not removing any thing from its past

→ Hence as $C^{(t)} \uparrow$ $O^{(t)} = \tanh(C^{(t)}) \Rightarrow$ Saturates due to large values

This saturation again will cause ~~grad~~ learnability problems in LSTM

We can use a constant decay factor, but problem is we have

$$C^{(t)} = \underbrace{0.9}_{\text{Remember only 90\% of history}} * C^{(t-1)} + \text{Ingate} * \tilde{C}^{(t)}$$

To remember things only that matter to us selectively, Constant factor forgets everything over time and doesn't discriminate

Ex:- Remember "Black pearl"
Forget "is a"

→ Use a dynamic fuzzy gate that is learnable and forget selectively the irrelevant info

$$C^{(t)} = \text{Forget gate} * C^{(t-1)} + \text{Ingate} * \tilde{C}^{(t)}$$

→ Also Forget gates are initialized to values closer to one as at the beginning it has to remember everything and slowly decrease value by learning to remember selectively

1) Difference between sequence → fixed size & fixed size → sequence

Sequence To fixed : kind of similar to RNN encoder which summarizes all the info from arbitrary length input to fixed size value

Ex: Text summarization, Embedding

Fixed to sequence : Like to Decoder of RNN which expands on given summary dense info and creates an arbitrary length o/p

Ex: Image captioning

Attention and Memory

1) Attention and context vector

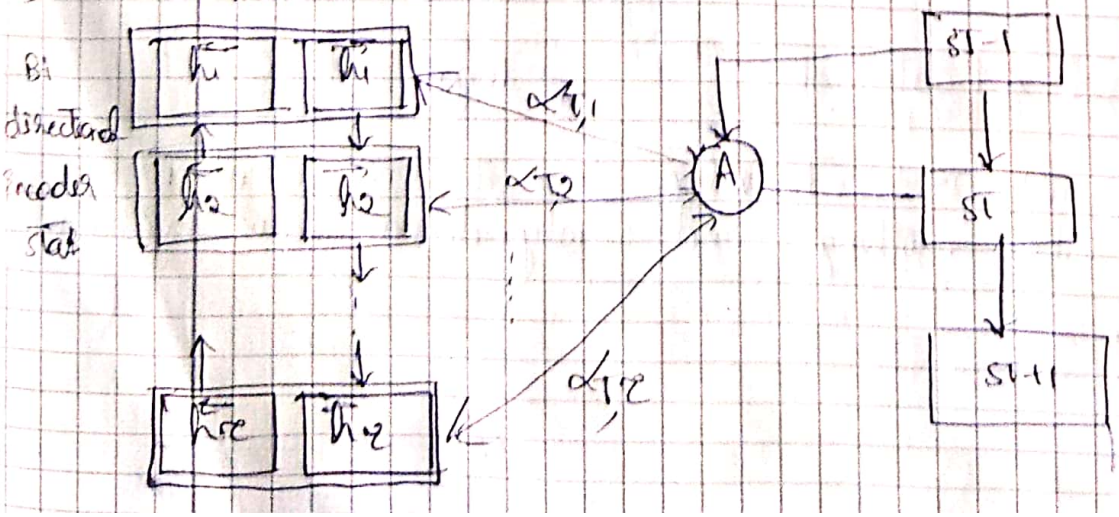
→ Attention is a mechanism of selectively looking at some states based on a similarity score between each state v/s previous o/p state

→ Context vector consumes relevant info from input/output states of arbitrary length and summarizes them into a fixed size rich high dim vector

2) Soft Attention
deterministic
exact gradients
O(input size)
easy to train

Hard Attention
stochastic
approximated gradient
O(1)
harder to train

3) Attention mechanism



$$\sum_{i=1}^{T} \alpha_{t,i} = 1 \quad \Rightarrow \quad \alpha_{t,i} = \frac{e^{a_{t,i}}}{\sum_{k=i}^{T} e^{a_{t,k}}} \quad -(1)$$

$$a_{t,i} = f(s_{t-1}, h_i, \theta) \quad -(2)$$

③ - additive attention :- $a_{t,i} = \underbrace{V^T}_{\text{scale}} \tanh(W h_i + U s_{t-1} + b)$

④ - Dot product attention :- $a_{t,i} = \vec{h_i} \cdot \vec{s_{t-1}}$

⑤ - Scaled Dot product attention :- $a_{t,i} = \frac{\vec{h_i} \cdot \vec{s_{t-1}}}{\text{scaling factor}}$

→ Attention mechanism is primarily introduced to remove bottleneck context vector in Encoder-Decoder architecture.

→ Attention enables parallel access to all encoder states and dynamically compute relevant summary at each time step depending on previous decoder & all encoders.

→ This also means that path for gradients is shortened and to encoder states.

→ In the above we can see that encoder states $\vec{h_1} - \vec{h_2}$ is accessed selectively by decoder depending on its need

→ How much to access from each encoder at single time step?
We use a softmax based mask over encoder, and access the amount of info from each encoder as given by mask value (1)

→ How is the mask calculated?
Use a similarity measure b/w each encoder state and previous decoder state ②, ③, ④, ⑤

④ How could adding attention mechanism into char RNN improve generated texts?

→ Direct access of character generated at 't' with characters at '0 - t-1' makes of production much more refined as we don't only look at previous memory state, but all past memory states

→ Gradient paths could decrease due to direct connection and generation could be made almost in parallel

5) Advantages of external memory and Disadvantages

→ Network doesn't learn the distribution of data or patterns in data, instead they learn when and how to access these patterns already stored externally. Hence Network doesn't need to store weights/biases pertaining to data which saves need for huge compute resources

→ Patterns or underlying parameters could be swapped inside similar domain and access pattern of controller could be changed to attune to new patterns with relative ease & less training. (Swapping Authorship graph parameters with citation n/w parameters)

→ Typically to make process differentiable we access all memory cells by weighted mask. This can lead to potentially large access times for huge external memory

→ Since it's differentiable cells store real values and gradients are real, Due to large memory chances are gradients could explode/vanish making learning really slow

→ Cannot take advantage of GPUs as external memory is a RAM or Harddisk

6) Types of accessing memory

Content based Addressing : We pick selective percentage of all memory cells to address based on contents stored inside the memory cells. This is similar to information retrieval. We provide a query vector whose similarity is compared to each row vector of Memory. Most similar ones get high score and others get low scores

$SI = Sim(q, M_t(i)) \equiv \vec{q} \cdot \vec{M_t(i)}$ or any other sim measure

→ This similarity score is softmax normalized and used as a mask to access memory

$$Softmax(SI) = W = \frac{e_i^{SIM}}{\sum_K e^{SIM}_K}$$

→ We usually use a scaler β to sharpen softmax so we can read small bit of info from small no of locations

$$\text{Softmax}(\text{SIM})_i = \frac{e^{\beta \text{Sim}_i}}{\sum_k e^{\beta \text{Sim}_k}}$$

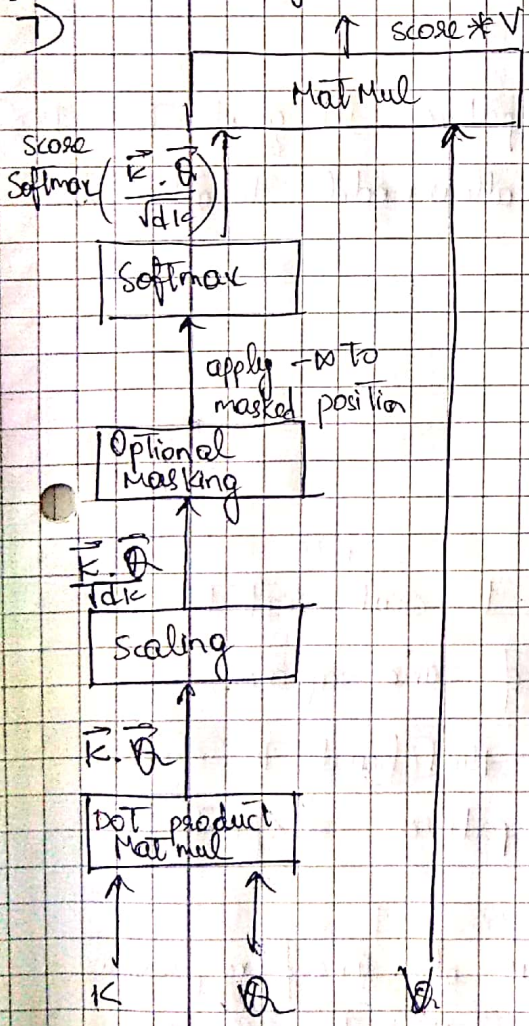
Interpolation addressing :- Goal of like Youtube recommendation, where current o/p addressing is based on previously addressed Memory

$$M_i(t) = g_i \bar{M}_i(t) + (1 - g_i(t)) M_i(t-1)$$

gate

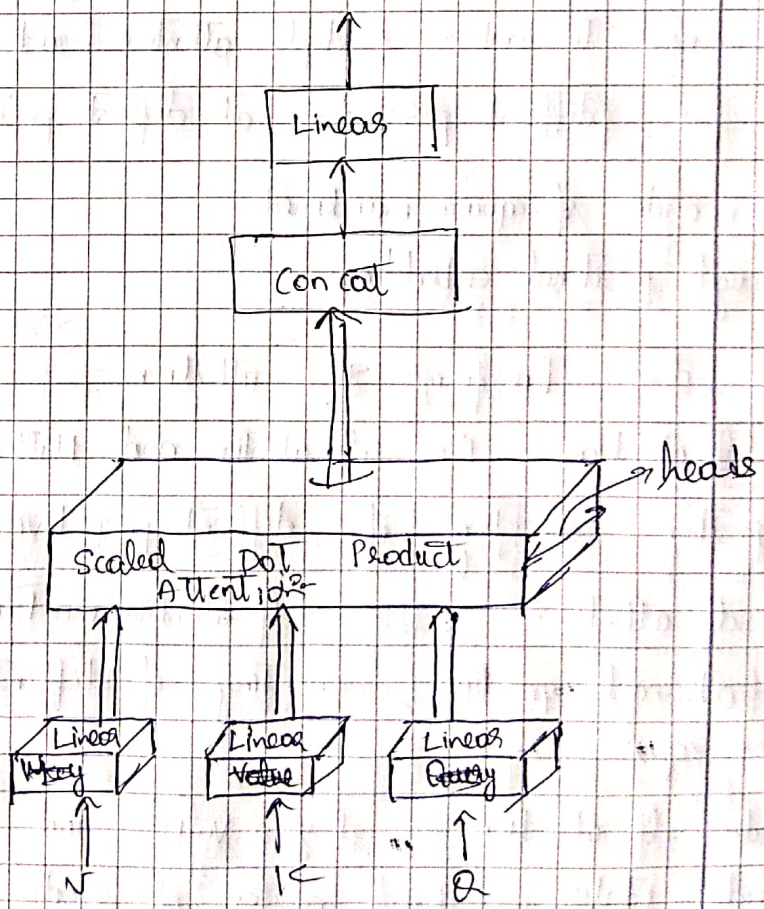
Location based addressing :- Address locations based on position rather than contents of memory itself. Similar to choosing of pointers rather than contents

1)



Transformers labelling

Multihead attention



8) How is parallelization achieved in transformers

→ It is achieved first by using a self attention layer with shared params for each token of sequence

→ Next through point wise feed forward n/w which applies same transform to all incoming values ||ely

9) Self-Attention v/s Multihead attention

Multihead attention :- $\text{Multihead}(Q, K, V) = (\text{concat}(\text{head}_1 \dots \text{head}_8))W^O$

$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

Multiple attention models (be it self or inter) is called Multihead attention

Self attention :- A single attention head where each token tries to encode itself through the lenses of its neighbours in the same sequence

10) Advantage of Multihead attention

→ Each attention head focuses on different perspectives like gender, time, pose. Hence we need to combine multiple attention heads to allow model to attend to info from different perspectives at different positions

11) How is order of sequence maintained

→ Through positional embeddings

12) How does Transformer use attention

Encoder self attention :- Uses self attention and Multihead to create context at each position consisting of different perspectives of same sequence

Decoder self attention :- Uses self attention, masking & Multiheads to create a context at each position, consisting of different perspectives of past states of same sequence

Encoder Self attention :- keys, values and queries all come from previous encoder states. Each position of encoder can attend to all states positions (past, current & future) of previous encoder

Decoder Self attention :- Key, Query & Values obtained from previous decoder states

Attends to only till past + current positions of sequence from decoder (auto regressive)

Other future states are masked by setting them to $-\infty$

Encoder - Decoder attention :- Takes query from ~~previous~~ current decoder self attention and key, value from topmost encoder. Attend to all positions of encoder states using decoder query & extract content

12) Connection b/w encoder - decoder is established in decoder's intra attention module where we take & use only top most encoders K, V for each decoder modules Query

13) Final layer of Transformer is a linear + Softmax combo over entire vocabulary. Predicts next probable token from vocabulary

Language models

1) Definition:- Task of assigning prob distribution over sequence of words that matches distro of language

2) Flavors of Word2vec

Skipgram:- given a word predict its neighbours / content

s

Category	Skipgram	CBOW
Training Time	comparitively ~~less~~ more as Training examples are more	Lesser to Skipgram
Data set	Can work well on smaller data as it can generate more example	Need to have a bit more data than Skipgram
Task	given a middleword, find its neighbours / context	given multiple neighbours / context predict single word
Prediction task app	Could be used as contextual embedding	Can mostly be used for filling the blanks (Auto complete) kind of apps
	Can predict more words	Most likely doesn't predict more words

3) Problems with loss function

→ Instead of finding most probable word given sequence using softmax Try to maximize a score for correct sequence word & penalty for wrong word

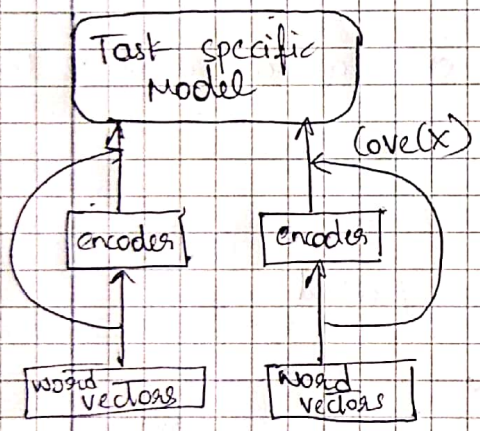
$$\sum_i \sum_V (1 - X_{correct} + X_{wrong})$$

4) Output bottlenecks:- softmax should be computed for all classes which takes lot of time, instead use -ve sampling or hierarchical softmax

5) Language Models – Contextual

Limitations i) Pretraining on supervised Translation task → available data

ii) Contribution To performance depends on task specific model



ELMO :- Unsupervised pretraining of stacked bidirectional LSTM n/w

→ Trained

$$v_i = \gamma \sum_i s^i h_i$$

(γ: scale, s^i: softmax)

GPU for deep learning

→ Memory Bandwidth :- Can move around more memory than CPU

→ Thread Parallelism :- Even though we move around lots of memory due To lots of threads we can move large amount of stuff which hides latency

→ Large and fast L1 cache & registers that are tunable & programmable