Neural Networks in NLP

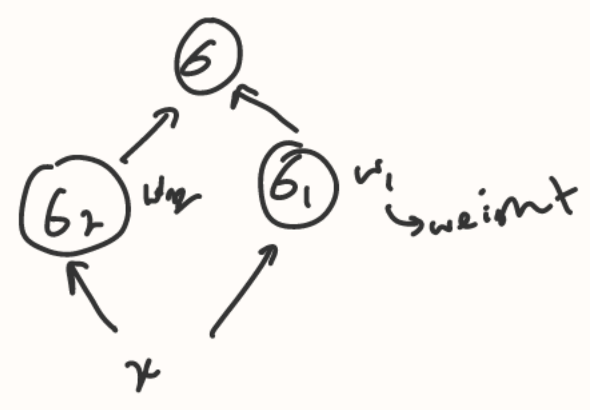
Classify tweets based on no. of chars

w_1 w_2

$2\sigma_2 - 1\sigma_1$ → Probability > 1

<u>soln</u> Bind whole soln into another sigmoid

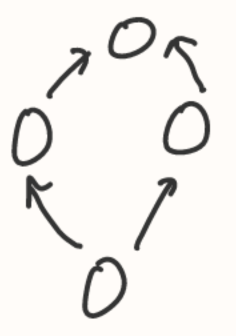
Goal of sigmoid : betwn 0 and 1



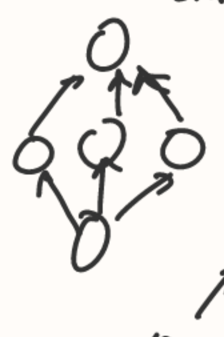
Logistic Regr : $y = \sigma(wx+b)$

↓

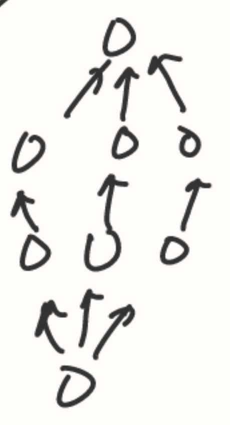
Neural network : $y=$



increase nodes →



increase layers →



Fully connected – neural network need not be

feed forward : uniform direction of calculations

| NN need not be

prev weights → learn next weights

Recurrent neural networks

| LR – linear prob – curve descending – global minima
| Guaranteed to find a soln.

Here, instead of global minima, we have a local minima

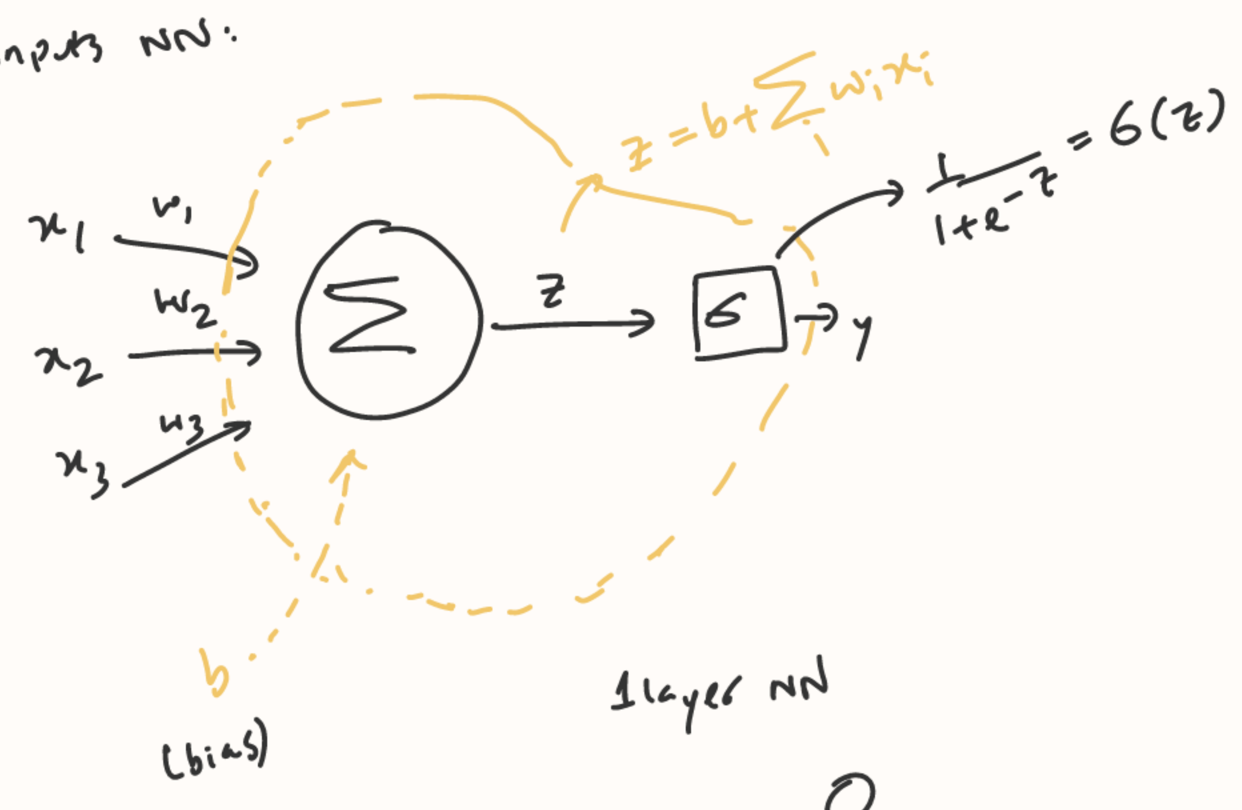
Not guaranteed to have a soln

<u>soln:</u> defensive technique (to avoid local minima)

– Random initialization

weight values in NN
(prev. weight value 0)

3 inputs NN :



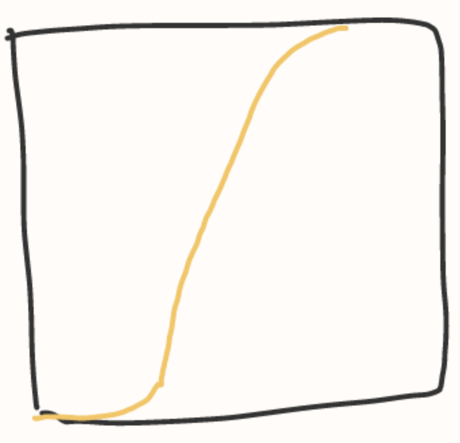
1 layer NN

single unit → 1 node O

perceptron : outputs a binary outcome (0 or 1)
linear activation function
(sometimes non linear)

for last layer | Sigmoid activation func. → binary layer → non linear A.F.
| Softmax for multinomial classification

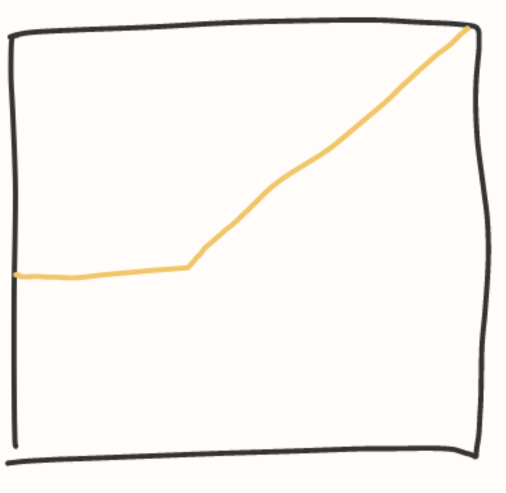
for hidden layer | tanh
| Relu



$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

tanh

more suitable to handle outliers
(projects outliers to mean value)



$y = relu(z) = max(z, 0)$

Rectified linear

–ve value → will be 0

> 0 → that value kept

→ handle outliers

prob suppress high values of z to 1

Apply σ or tanh for high values of z → 1

vanishing gradient (gradient becomes 0) → differential = 0

relu keeps same values of z
gradient doesn't become 0

output layer → softmax

Neural Network (NN)
tanh and RELU
Perceptrons

XOR problem
soln: use intermediate layers
Feed forward NN — output of one layer as input of next layer

Multinomial Logistic Regression — softmax
(MLR)

Sigmoid	softmax
Binary classification	multinomial classification

1 layer

softmax → output y is a vector

probability

1st class

$$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.1 \end{bmatrix} \begin{bmatrix} 0.02 \\ 0.08 \\ 0.9 \end{bmatrix} \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix}$$

probability that this output is from 2nd class

3rd class

2 layers

output, hidden layer

multilayer NN == MLR

- add as many hidden layers as possible | not possible to define as many

- tanh or RELU in intermediate layers — softmax/sigmoid (don't use tanh or RELU)

-

Why do you use hidden layers?

You try many diff combinations of features to classify
Hidden layer used for feature representation
which feature contribute more to classification

multilayer Notation

$z^{[1]}$ output after 1st layer

$a^{[0]}$ input
g activation function
b bias

// Necessity of non linear Activation function
tanh, RELU, softmax ...

If g1, g2 both linear:

$$z^1 = w^1 x + b^1$$
$$z^2 = w^2 z^1 + b^2$$
$$= w^2 (w_1 x + b^1) + b^2$$
$$= \underbrace{(w^2 w^1)}_{w'} x + \underbrace{((w^2 b^1) + b^2)}_{b'}$$
$$= w' x + b'$$

g

basically 1st layer (no feature learning happening)

Non linear Af activates feature representation
w/o them, no trying out combo of features
hidden layers don't learn anything

Replacing the bias unit

x features
b bias → $x_0 = 1$

prev | +1 (last layer last node)
new

Text classification
Lang modeling

// Why feeding handcrafted features?

we learnt embedding of each token
bert, word2vec, glove — trained embeddings — outside source

Pooling: combining all embeddings, maybe learn a new embedding

$$\begin{bmatrix} 0.55 \\ -0.9 \\ 0.077 \end{bmatrix} \begin{bmatrix} \\ \\ \\ \end{bmatrix} \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

desert way great

$\oplus$ $\begin{bmatrix} - \\ - \\ - \end{bmatrix}$ → max of row 1

$\frac{\cdot + \cdot + \cdot}{3}$

which pooling method to use? — Based on empirical evidence

Pretrained / transfer learning : Learning from an already learned embedding
\ using learning from another source

units in NN

Issue: texts come in diff sizes

Neural LM

LM Predicting next tokens based on prev tokens
↓
Lang. modeling

// Transformer based LM / better than n-gram based LM
Neural LM

Embedding for each token from outside source

unknown sample in Ngram LM → cannot predict a word that didn't appear in the training sample
(word not in training corpus)

In neural LM, not mandatory for a word to be in training sample.
Similar tokens – cat and dog.
what goes for cat, goes for dog.
can predict

Training
I ate English breakfast
I had good lunch at a chinese restaurant.

Testing
Lunches are good in ______ restaurant. ✓ can predict

Lunches are good in ______ cafe. ✗ cannot predict

Neural LM knows restaurant and cafe similar. → can predict.

Output layer : $z_2 = u \times a_1$

Hidden layer (RELU A.F) $a_1 = g(z_1)$

Embedding layer $w \cdot e + b$

Eqns

$$e = [Ex_{t-3} ; Ex_{t-2} ; Ex_{t-1}]$$

$$h = \sigma(we + b)$$

$$z = uh$$

$$\hat{y} = \text{softmax}(z)$$

noise / word not in pretrain corpus → return noise/random embedding

2 layer Network

Every layer has weights
update weights based on loss function
depends on output layer | loss function only on last layer

Backpropagation : calculate weight of first layer

computation Graphs
↓
Eqn to graph

forward pass

Backward pass

change of a impacts L? $\frac{dL}{da} = \frac{dL}{de} \cdot \frac{de}{da}$

Training LLM

large corpus, gold label? ✗
↓
unsupervised learning approach
(not annotated)

sliding window
self supervised learning
Train your model to predict next work

BERT Paper
NSP (next sentence Prediction) – minimize loss function
MLM (mask Lang modeling)

lunches are <TOK> in restaurants.
↓
token

unfreeze embedding → modern LM
(update embedding layer)

LN 7 done

RNNs and LSTMs

Recurrent Neural Networks

--- dash line on h_t means Hidden layer dependent on sth.

like feedforward Neural Network

Diff betwn RNN and feedforward NN:

non-linear (cycles in network) dependency

linear representation of network

Training an RNN LM → Autoregressive generation — Automatically start predicting (Don't specify gold label)

Self-supervision

Teacher forcing → use predicted output to calculate loss

Sampling technique: Give prev words, ask LM to predict next word

Text generation/LM

If it can generate an accurate token, loss is minimal

word2vec, fasttext → outside source

RNN for sequence labeling

Each token of sequence has a gold label

POS-tagging (parts of speech)

RNN for sequence classification

I love to play cricket — Positive (overall)

no need for every token

n - last token

map output to 3 classes — FFN (+, -, neutral)

prob of RNN: Loss of info by the time it comes to the last token

Autoregressive Generation with RNNs

Teacher forcing before — AG with RNNs — fine tuning method

(training method)

(Task) — text gen

Soln Average Pulling / max Pulling

Bidirectional RNN → RNN1, RNN2

Stacked RNN → RNN1, RNN2, RNN3

more layers → learn more abstract/sophisticated features

Limitations RNN

- Info passed is local
(Does not perform well for distant info)

| Provide info in current context
| Carrying info from forward

- vanishing gradient problem

back propagation : past info depending on future info
(backward pass)

Remember important part, forget rest

↓

LSTM
removes info not needed
add info which might be used to take future decisions
use Gates

Focus on imp parts of a sentence → Attention

- feed forward layer
- sigmoid activation function → output 0/1 (0 → erase, 1 → remember)
-

Hadamard Product : Element wise multiplication
↳ A collection of dot products

Forget Gate : forgets info not needed
binary masking (sigmoid function)

higher weights to info for keeping

Info calculated for current context → use tanh activation func.
Keeps weights as it is

>0 → as it is info
<0 → 0
(lose less info)

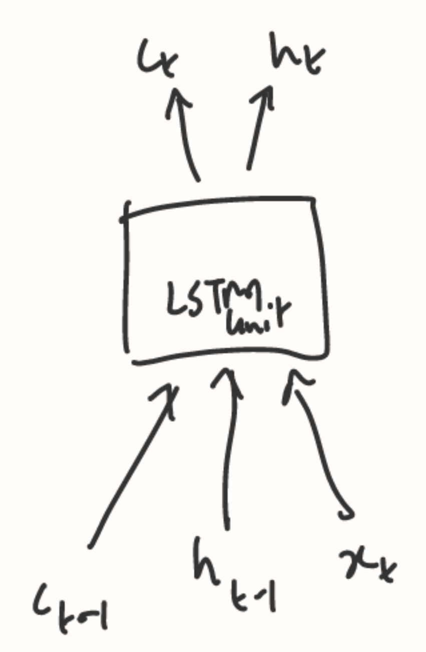
Add Gate

info to be retained
(+) info to be erased

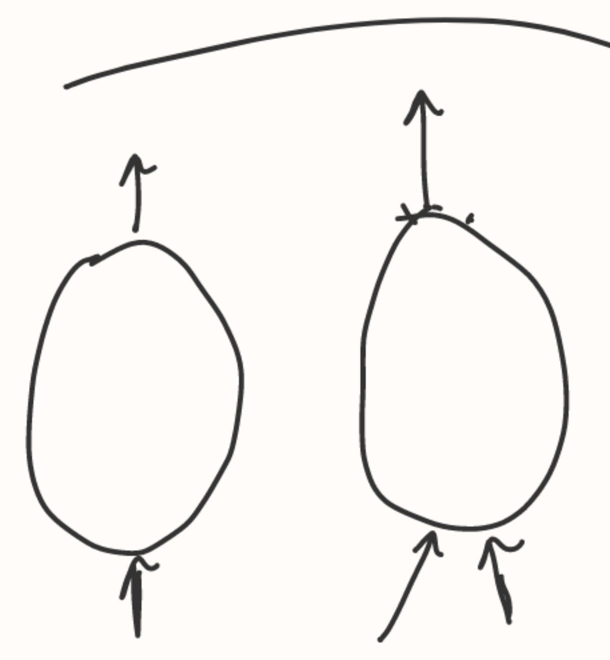
updated info (Next context)

Output Gate
- context with updated info
- hidden layer calculated using prev hidden layer + current input + current context

| Feed forward neural network
| RNN – knows most recent info
| LSTM – most relevant info/context

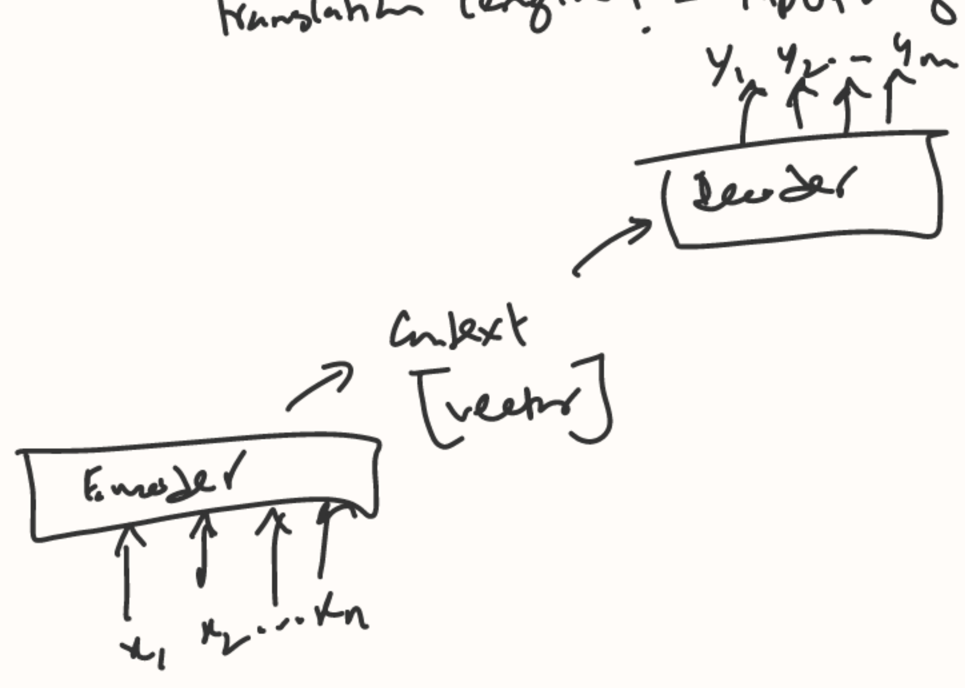


Activation function is basically used for decision making
0/1



RNN NLP Architectures
1. Sequence labeling task
2. Sequence classification
3. Language modelling
4. Encoder-Decoder

seq2seq modeling
translation length != input length



Translation task

Inference == prediction

Training the encoder/decoder model

Sample wise loss (not element wise loss)

| training – loss – weight submit
| validation set – loss calculate [illegible] for [illegible]?

Decoder
~~Hidden layer~~: $h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)$

$c = h_n^e = h_0^d$

$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c)$

$z_t = f(h_t^d)$

$$y_t = \mathrm{softmax}(z_x)$$

LSTM performs uniformly better than RNNs.

Ch10 : Transformers and LLMs

30K-1lac vocab : Age 30 person
740 words learn everyday
reading enriches vocab

Distributional hypothesis - words that appear together have a similar meaning

let Lm read books
RNN, LSTM, Encoder-Decoder
Those architectures cannot learn complex association of words
Soln : Transformers

LSTM does sth like attention

Building block : Attention - to focus on the relevant part of the sentence
Attention mechanism -
self attention - Building block of transformers

What part of RNN does LSTM improve?
Remember distant info by forcefully adding some info

LSTM also not accurate (probabilistic model)
could forget

uncertainties in LM

The notion of letting LM read and learn association of words : Pre-training
(filling LM with info)

Downstream task : Fine-tuning

To make contextual representation of each word based on its surroundings - word embedding
(don't use frequency based methods like tf-idf)

Paralization ? Parallelization

Transformer
- Linear layer
- Feed forward network
- self-attention network

How self attention is different from attention?
hidden layers not connected to each other (no recurrent connection)
compare each of the word (not the hidden layers)
compare hidden unit

Backward looking self att: (only look at preceding words) ← . . . . . it
↓ can do
Autoregressive generation

Bidirectional looking self att : (look all words)
Not dependent on each other

dot products

Create a representation of context for each word based on surrounding words - self att.

How to calculate a3 ?
1st step : $score(x_i, x_j) = x_i \cdot x_j$

2nd step : $\alpha_{ij} = softmax(score(x_i, x_j))$
↑ att. weight

3rd step : $a_i = \sum_{j \leq i} \alpha_{ij} x_j$

actual input 끼리 att. calculate
(in att. hidden input 끼리 att. calculate)

which word is more relevant → give more weights

prev: outsource embeddings.
Transformer have their own embeddings.

Query , key , value - 3 roles of self att

Each token multiplied with softmax

A cow will come.
key · focus/query · value

Each input has these 3 representations
(x_1, x_2, x_3) · (q, k, v)

steps of self att.
1) Comparison : compare query of x_3 with key of x_1 - Dot product
key of x_2 - "
key of x_3 - "

2) Normalize : send this dot product to softmax activation function

3) calculate value : multiply each vector with their corresponding softmax value

$\underbrace{\text{vector}}_{\text{hidden layer}}$ … $\underbrace{\text{softmax value}}_{\text{weight}}$

Add → Get a single vector

↓

Output of self-att.

Dimension

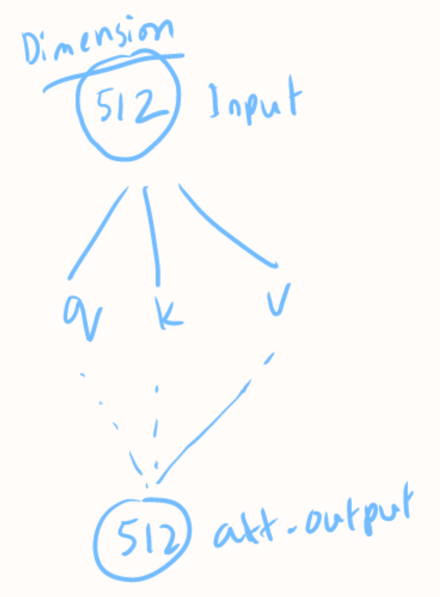
$d_k = d_q = 64$

Each word represented by 64 values

A

↓

$[\ \underbrace{\ldots\ldots\ldots}_{64 \text{ values}}\ ]$



Transformer remember long context — very good autoregressive generator

RNN and LSTM cannot look at very distant info — fail after a few tokens

Mask out the future → $-\infty$

(Lower A matrix)

1 self att unit

// A cat is chasing a rat.

Syntax relation

Semantic relation

Multihead att.

Transformer Blocks

Residual connection : original info pass to the next layer

layer norm : to keep values in a calculable range

$\left.\begin{array}{l}\gamma \text{ - gain} \\ \beta \text{ - offset}\end{array}\right\}$ learnable curve

" 96 transformer blocks inside GPT3 "

cat $\boxed{\text{sat}}$ on mat

$q_{cat} = [0.2\ \ 0.3\ \ 0.45\ \ 0.35]$

$k_{cat} = [\qquad \text{"} \qquad]$

$v_{cat} = [\qquad \text{"} \qquad]$

$q_{sat} \cdot k_{cat} = 3.45$

$q_{sat} \cdot k_{sat} = 120$

$\alpha_{21} = \dfrac{3.45}{3.45 + 120} \approx 0.45$

$\alpha_{22} = \dfrac{120}{3.45 + 120} \approx 0.55$

attention weight value

$a = \sum \alpha_{ij} v_j$

$a_2 = (0.45 \times v_{cat}) + (0.55 \times v_{sat})$

Post-norm vs prenorm transformer

↓

traditional

layer normalization w/ raw input ([illegible])

$\left.\begin{array}{l}\text{word embedding} \\ \text{and position embedding}\end{array}\right\}$ for each token → To prevent exchange of position which can change meaning of the sentence

David beat John w/ a stick

token exchange

John beat David w/ a stick.

positional embedding used in negative sentiment

NOT _ good → negative sentiment

Language modeling Head

↓

Added on top of transformer block to match dimension

vocab size = no. of tokens in dataset = # cols of embeddings

LM = Next token/word prediction task

(Lang. modeling)

Transformer has a long context window.

LLMs w/ transformer
|
LM trained (generalized)

Sentiment Analysis
Ques-Answering
Summarization
} Downstream task solver

RAG - use external sources w/ LLM (to prevent hallucination)
↓
Retrieval Augmented Generation

- knowledge graphs
- vector database
- prompt engineering (chain of command
↓
Divide query into a chain

Toxic lang Generation - data filtering

Privacy Issues - remove private datasets from training data

Datasheets/model cards

FINALS

After mid

2 Ques from each 3 chapter

RNN, LSTM, Transformers

maths
Diagrams of underlying architecture

self supervised training

Dimension

$1 \times |V|$
$1 \times d$
$\Big\}\ d \times |V|$

eg