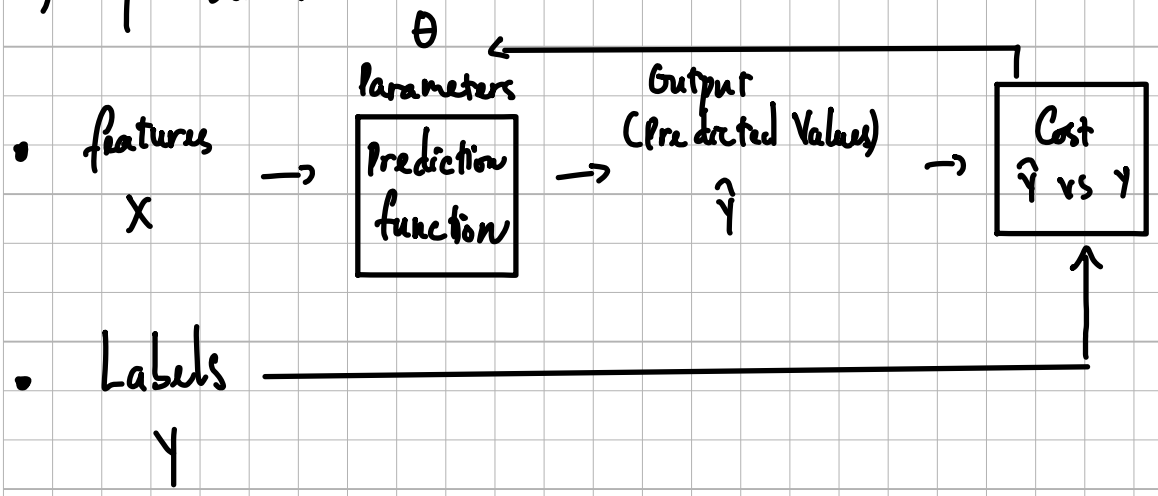
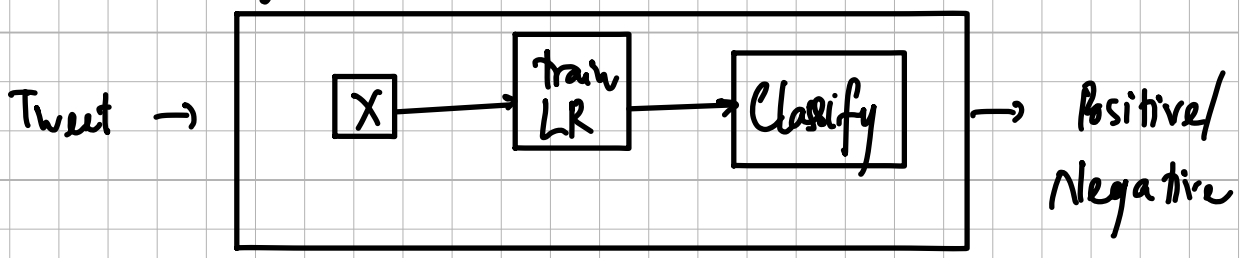
Coursera - NLP

(1) Classification and Vector Spaces

+) Supervised ML



+) Sentiment Analysis



+) Vocabulary

V = list of unique words $|V| = m$

+) Feature Extraction

Sparse representations → $\theta = [\theta_0 \dots \theta_m]$ (bias ↑)

+) Negative and Positive frequencies:

+) Feature Extraction with frequencies:

$X_m = [1, \sum_w freqs(w, 1), \sum_w freqs(w, 0)]$

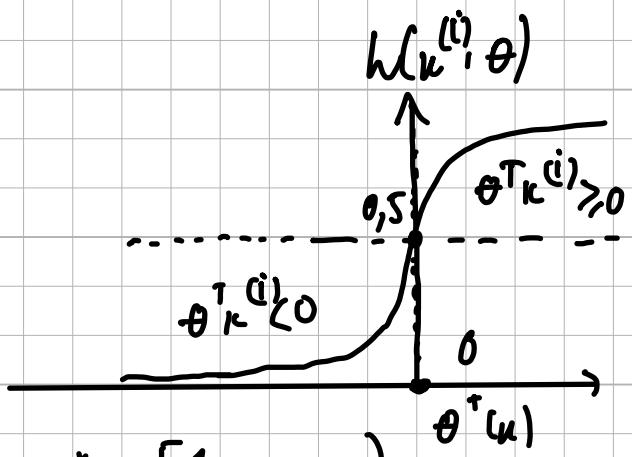
+) Stemming and Stop words

+) Putting it All Together

$X.shape = (m, 1)$

$$X = \begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} \end{bmatrix}$$

$$\begin{cases} \theta^T x^{(i)} \geq 0 \\ \theta^T x^{(i)} < 0 \end{cases}$$



$x = [1, pos, neg]$

+) Logictic Regression

$$h(x^{(i)}, \theta) = \frac{1}{1 + e^{-\theta^T x^{(i)}}}$$

+) Training

initialize $\theta \rightarrow h = h(X, \theta) \rightarrow \nabla = \frac{1}{m} X^T (h - y)$

↓

update $\theta = \theta - \alpha \nabla$

↓

Min Loss $J(\theta)$

Until good enough

+) Testing

$$\frac{\sum_{i=1}^{m} (pred^{(i)} == y_{val}^{(i)})}{m}$$

$y^{(i)} = 0 \quad \hat{y} = 0$

$y^{(i)} = 0 \quad \hat{y} = 1$

+) Cost function Intuition:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h(x^{(i)};\theta) + (1-y^{(i)})\log(1-h(x^{(i)},\theta))\right]$$

(2) Naive Bayes

+) Probabilities

→ to calculate a probability of a certain event happening

+) Bayes rule

Conditional Probabilities

$$P(\text{Positive}|\text{"happy"}) = P(\text{"happy"}|\text{Positive}) \cdot \frac{P(\text{Positive})}{P(\text{"happy"})}$$

$P(\text{"happy"} \wedge \text{Positive}) = 2\%$

$P(\text{"happy"}) = 13\%$

$P(\text{"positive"}) = 40\%$

$P(\text{Positive}|\text{"happy"}) = 977$

$$P(X|Y) = P(Y|X) \cdot \frac{P(X)}{P(Y)}$$

+) Naive Bayes for Sentiment Analysis

"Events are all independent → rarely the case"

+) Inference condition rule for binary Classification

$$\prod_{i=1}^{m} \frac{P(w_i|pos)}{P(w_i|neg)}$$

+) Laplacian smoothing ↓0 class ∈ {Positive, Negative}

$$P(w_i \mid class) = \frac{freq(w_i, class) + 1}{N_{class} + V_{class}}, \quad \to 0$$

N_{class}: freq of all words; V_{class}: # of unique words in class

+) log likelihood ≥0 ≤0

$$ratio(w_i) = \frac{P(w_i \mid pos)}{P(w_i \mid neg)}$$

Positive ∞ / Neutral 1 / Negative 0

$$\approx \frac{freq(w_i, 1) + 1}{freq(w_i, 0) + 0}$$

Naive Bayes's Reference

$$\Rightarrow \log\left(\frac{P(pos)}{P(neg)} \cdot \prod_{i=1}^{m} \frac{P(w_i \mid pos)}{P(w_i \mid neg)}\right) > 1$$

= log prior + log likelihood, $\in [-\infty, +\infty]$

Word Sentiment

$$ratio(w) = \frac{P(w \mid pos)}{P(w \mid neg)}$$

$$\lambda(w) = \log \frac{P(w \mid pos)}{P(w \mid neg)}$$

$$\sum_{i=1}^{m} \log \frac{P(w_i \mid pos)}{P(w_i \mid neg)} = \sum_{i=1}^{m} \lambda(w_i) \quad \begin{cases} > 0 \to \text{positive} \\ < 0 \to \text{negative} \end{cases}$$

+) Training Naive Bayes

Step 0: Collect and annotate corpus → Step 1: Preprocess

- lower case
- Remove punctuations, urls, names
- Remove stop words
- Stemming
- tokenize sentences

Step 2: Word count

freq(w, class),

Step 3: P(w|class)

$$\frac{freq(w, class) + 1}{N_{class} + V_{class}},$$

Step 4: Lambda

$$\lambda(w) = \log \frac{P(w|pos)}{P(w|neg)},$$

Step 5: Get the log prior

D_{pos} = # pos tweets

D_{neg} = # neg tweets

$$\log prior = \log \frac{D_{pos}}{D_{neg}},$$

(balanced!)

+) Confidence ellipse

+) Testing

- X_{val} Y_{val} ν log prior

  score = predict (X_{val}, ν, log prior)

  pred = score > 0

  $$\frac{1}{m} \sum_{i=1}^{m} (pred_i == Y_{val_i})$$

- Unseen = neutral words

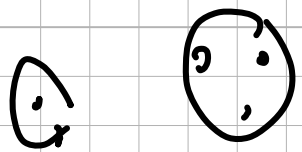
- Application of Naive Bayes
  - Sentiment analysis
  - Author identification
  - Information retrieval
  - Word disambiguation
  - Simple, fast and robust

+) Naive Bayes Assumptions
  - Independence
  - Relative frequencies in corpus

+) Error Analysis
  - Removing punctuation and stop words
  - Word order

- Adversarial attacks
  - Sarcasm, Irony and Euphemisms

(3) Vector Space Models



Different meaning vs. Same meaning

Capture dependencies between words,

- Information Extraction
- Machine translation
- Chatbots

fundamental concept

"You shall know a word by the company it keeps"

- neighboring words → center word
  - synonyms and antonyms can be easily interchanged,

+) Word by Word and Word by Doc

Co-occurrence matrix

W-by-W: # times occur together within a certain distance k

e.g.

	apple	raw	like	I
data	2	1	1	0

W-by-D: # times occurs within a certain category

e.g.

	Entertainment	Economy	ML
data	500	6620	9320
film	7000	4000	1000

measure of "similarity": Angle
Distance

axis = 0 ⇒ get for each column

axis = 1 → get for each row

+ Euclidean Distance

np.linalg.norm(v - w) ⇒ problems!

+ Cosine Similarity

when corpora are different sizes

cos = 0 ⇒ dissimilar $\theta = 90^\circ$

cos → 1 ⇒ similar

+ Manipulating Words in Vector Spaces ⇝ [ ]

+ Visualization and PCA

- Motivation

Original space → Uncorrelated features → Dimension reduction

- Visualization to see words relationships in the vector space.

+ PCA algo : Eigen values, Eigen vectors

Eigenvector: Uncorrelated features for your data

from covariance matrix

Eigenvalue: the amount of information retained by each feature

Mean Normalize Data

$$x_i = \frac{x_i - \mu_i}{\sigma_{x_i}}$$

↓

Get Covariance Matrix

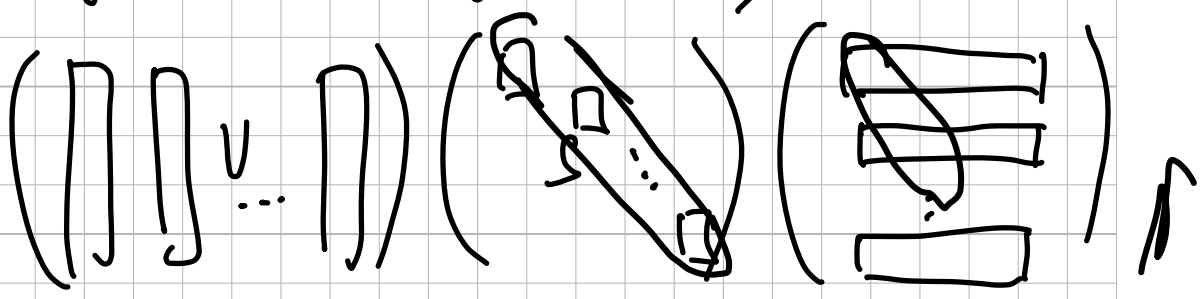
$$\Sigma$$

↓

Perform SVD

$$SVD(\Sigma)$$

- Eigen Vectors Eigenvalues → class order



Dot product to Project Data

$$X' = XU[:, 0:2]$$ → or n-dc

↓

Percentage of Retained Variance

$$\frac{\sum_{i=0}^{1} S_{ii}}{\sum_{i=0}^{d} S_{ii}}$$

Summary:
- Eigenvectors give the directions of uncorrelated features
- Eigenvalues are the variance of the new features
- Dot product gives the projection on uncorrelated features

(4) Machine Translation

Nearest neighbor search using locally sensitive hashing !

Machine Translation Document search

+ Transforming word vectors

Allign word vectors

$XR \approx Y$

→ train on the subset, predict the entire

Solving for R:

initialize R

in loop: → Norm 2

loss : $\|XR - Y\|_F$.

$g = \frac{d}{dR}$ loss gradient

$R = R - \alpha g$ update

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

np.sqrt(np.sum(np.square(A)))

Problem Set (5)29

Gradient

$$loss = \|XR - Y\|_F^2$$

$$g = \frac{d}{dR} loss = \frac{2}{m} (X^T (XR - Y))$$

+) K nearest neighbors : absolute values, closest

+) Hash table and hash functions

Hash function (vector) → Hash value

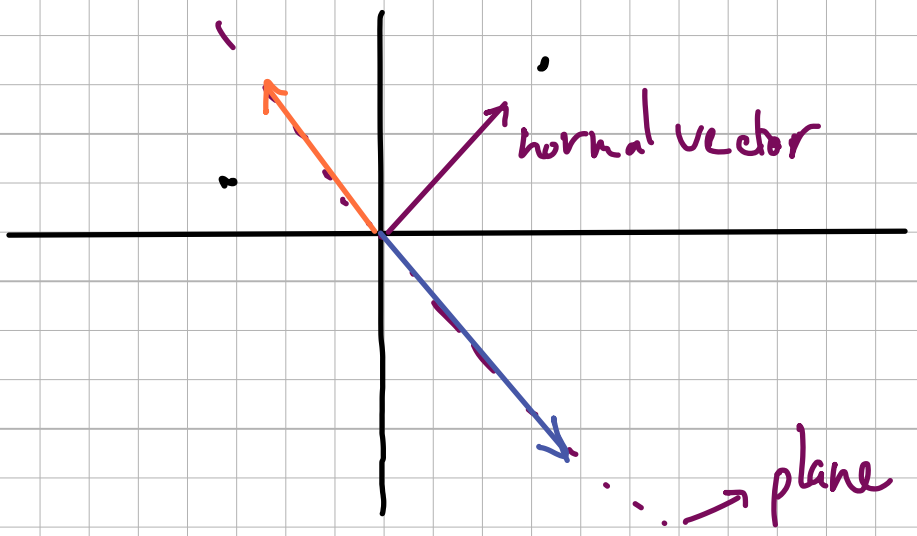
+) locality sensitive hashing

based on where located in vector space

n_buckets

{i: [] }

a line = a plane, parallel on the line



→ normal vector

def side_of_plane (P, V) → get relative pos to the plane)

+) Multiple planes → Dot products → hash values

sign$_i$ ≥ 0 → h_i : 1

sign$_i$ < 0 → h_i = 0

$$\text{hash} = \sum_i^{H} 2^i \cdot h_i \qquad G_i : [0 \ldots]$$

+) Approximate nearest neighbors

Radom planes

np. random . normal (size = (num_planes, num_dims))

+ Searching documents

doc : Σ word

Probabilistic models

(5) Autocorrect and Minimum Edit Distance

1) Autocorrect
   - Identify a misspelled word
   - Find strings n edit distance away
   - filter candidates
   - Calculate word probabilities

+ Building the model

Edit → Insert . $n = \overline{1,3}$
↓ Delete .
↓ swap 2 adj letters. (switch)
↓ change 1 to another. (replace)

Probabilities — # times the word appears

$P(w) = \frac{C(w)}{V}$ → total size of corpus (Corpus)

prob of word

+ Minimum edit distance (& switch)
  - Evaluate similarity between 2 strings
  - Find the minimum # of edits between 2 strings
  - Implement spelling correction, document similarity, machine translation, DNA sequencing and more

$D[i,j]$ = source $[:i]$ → target $[:j]$.

$D[i,j]$:

$$\min \begin{cases} D[i-1,j] + \text{del\_cost}, \\ D[i, j-1] + \text{ins\_cost}, \\ D[i-1,j-1] + \begin{cases} \text{rep\_cost}; \text{if } src[i] \neq tar[j] \\ 0; \text{if } src[i] == tar[j] \end{cases} \end{cases}$$

- Levenshtein distance
- BackTrace
- DP

	#	h	y	e
#	0	1	2	3
p	1	2	3	4
i	2	3	4	
e	3	4	5	

(6) Part of Speech tagging

+) Part of speech refers to category of words, lexical terms in the language : adv, v, n, adj...

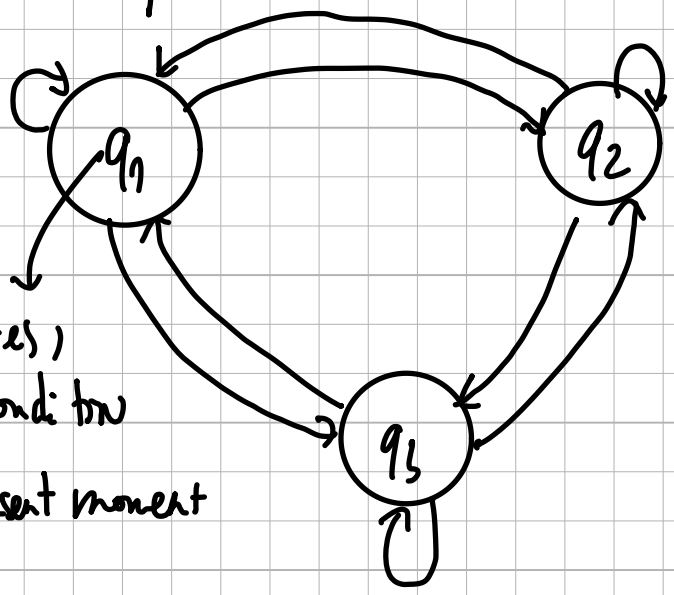
lexical term	tag	
noun	NN	
verb	VB	
determiner	DT	the, a
w-adverb	WRB	why, where

+) Applications → probs of POS tags

Named entities Co-reference resolution Speech recog

+ Markov chains : transition probs, states

Visual representation



$Q = \{q_1, q_2, q_3\}$

q_1 q_2 q_3

States)
certain condition
of the present moment

+) POS tags as states !

Markov property → simple, only depends on previous

Transition probabilities (matrix) ↗ # states

$(N \times N)$

current state ↖	NN	VB	O	
A = NN	0,2	0,2	0,6	
VB	0,4	0,3	0,1	
O (other)	0,2	0,3	0,5	

$$\sum_{i=1}^{N} a_{ij} = 1 \quad \text{(each row)}$$

$\begin{pmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{pmatrix}$

$(N+1) \times N$

! No previous words → initial state

π (initial)	0,4	0,1	0,5

+) Hidden Markov Models → decode hidden states

emission probs $N \times V$

↳ probability to go from POS tag to a specific word

+) Calculate Probs

• Transition

1. Count occurrences of tag pairs. → $C(t_{i-1}, t_i)$

2. $$P(t_i \mid t_{i-1}) = \frac{C(t_{i-1}, t_i)}{\sum_{j=1}^{N} C(t_{i-1}, t_j)}$$

Preparation of the corpus

<s> lowercase-insensitive punctuation

<s> . . .

• Smoothing

$$P(t_i \mid t_{i-1}) = \frac{C(t_{i-1}, t_i) + \epsilon}{\sum_{j=1}^{N} C(t_{i-1}, t_j) + N * \epsilon}$$

small value

$$P(t_{col} \mid t_{row}) = \frac{C(t_{row}, t_{col}) + 0{,}001}{\sum_{col=1}^{N} C(t_{row}, t_{col}) + N + 0{,}001}$$

+) Emission

You: 2

: 3 → Emission prob : You = 2/5

in a ...

You eat
the oatmeal
You eat

NN
VB
J

$$P(w_i \mid t_i) = \frac{C(t_i, w_i) + \epsilon}{C(t_i) + N + \epsilon}$$

$$\sum_{j=1}^{N} C(t_i, w_j)$$

+) Viterbi Algorithm - graph algo

1. Initialization step
2. forward pass
3. backward pass

prob max each path, keep track path end up taking
C D

Initialization

(1) ⇒ C prob of every word belongs to certain POS

w_1: $C_{i,1} = \pi_i * b_{i,\text{col index}(w_1)}$

$= a_{1,i} * b_{i,\text{col index}(w_1)}$

(2) ⇒ D: store different states you are going through when finding most likely pos tags for the given sequence words $w_1 \cdots w_K$

$d_{i,1} = 0$

t) Forward step $p(w_{j-1} \in t_k) * p(t_i | t_k) * p(w_j | t_i)$

$C_{ij} = \max_k C_{k,j-1} * a_{k,i} * b_{i,\text{col index}(w_j)}$

$d_{ij} = \arg\max_k C_{k,j-1} * a_{k,i} * b_{i,\text{col index}(w_j)}$

+ Backward pass

- Index start with 0!
- $\log(C_{i,j}) = \max_k \log(C_{k,j-1}) + \log(a_{k,i}) + \log(b_{i,\text{col index}(w_j)})$

$S = \arg\max_i C_{i,L}$ = index row D (save path) ⇒ and then use D

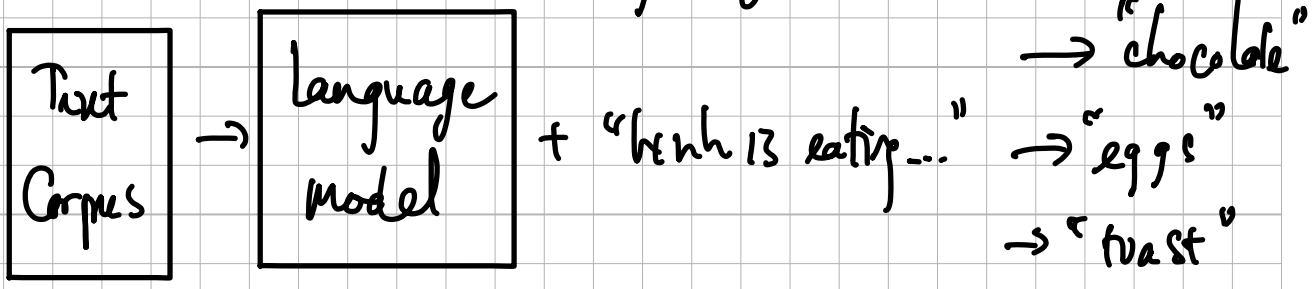
(7) Auto Complete

physical talk
↑ or sign

+) N-grams:

Speech recog, spelling correction, augmentative communication, ...

- Creating language model (LM) from text corpus is:
  - -) Estimate probability of word sentences
  - -) Estimate probability of a word following a sequence of words
- Apply this concept to autocomplete a sentence with most likely suggestions



Sentence auto-complete:
- Process a text corpus to N-gram LM
- Handle out of vocab words
- Implement smoothing for previously unseen N-grams
- LM evaluation

+) N-grams and probabilities (conditional prob)

↓

Set of words, punctuation treated like words

Corpus: I am happy because I am learning

Unigrams: {I, am, happy, because, learning}

Bigrams: {I am, am happy, happy because, ...}

Trigrams: {I am happy, am happy because, ...}

- Sequence notation:

$w_1 \dots w_m$

$w_1^m = w_1 w_2 \dots w_m$

$w_1^3 = w_1 w_2 w_3$

$w_{m-2}^m = w_{m-2} w_{m-1} w_m$

probability

- Unigram: $P(w) = \frac{C(w)}{m}$
- Bigram: $P(y|x) = \frac{C(xy)}{\sum_w C(xw)} = \frac{C(xy)}{C(x)}$
- Trigram: $P(w_3|w_1^2) = \frac{C(w_1^3)}{C(w_1^2)} = \frac{C(w_1^3)}{C(w_1^2)}$
- N-gram: $P(w_N|w_1^{N-1}) = \frac{C(w_1^N)}{C(w_1^{N-1})}$

→) Sequence Probabilities → A appears just before B

$P(A, B) = P(A) \cdot P(B|A)$

$P(A, B, C, D) = P(A) \cdot P(B|A) \cdot P(C|A,B) \cdot P(D|A,B,C)$

- Probability of a sequence:
  P(the teacher drinks tea) $\overset{\text{chain rule}}{=}$ ...

- Sentence not in corpus

→ Approximation:

- the Markov assumption indicates that only the last word matters
- Bigram:

$$P(w_n | w_1^{n-1}) \approx P(w_n | w_{n-1})$$

end − start + 1 = N − 1

Entire sentence

$$P(w_1^n) \approx \prod_{i=1}^{n} P(w_i | w_{i-1})$$

$$\approx P(w_1) \cdot P(w_2|w_1) \cdots P(w_n|w_{n-1})$$

- N gram

$$P(w_n | w_1^{n-1}) \approx P(w_n | w_{n-N+1}^{n-1})$$

+) Start and ending sentence

<s>

</s>

• N-gram model : add N-1 start token <s>
at the beginning of each sentence

just add one </s>

+ Calculate count matrix → prob matrix → LM

$$P(w_n \mid w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}, w_n)}{C(w_{n-N+1}^{n-1})}$$

→ Count matrix

Rows : unique corpus (N-1)-grams

Columns : unique corpus words

$C(w_{n-N+1}^{n-1})$ → sum(row)

• Bigram

count matrix → Sliding window 2-size

→ Probability matrix : cell / row(sum)

• Language model

- prob matrix => LM

+ sentence prob

+ next word prediction

- Log prob

$$\log(ab) = \log(a) + \log(b)$$

Algorithm:

1. Choose sentence start (higher value, randomly)
2. Choose next bigram starting with previous word
3. Continue until </s> is picked

+) LM evaluation

- Train / validation / test split

	Smaller corpora	Large corpora (typical for text)
Calculate loss ⇐	80% train	98% train
Finetune ⇐	10% validation	1% validation
hyperparam evaluate ⇐	10% test	1% test

test data - split method

- Continuous text
- Random short sequences

+) perplexity metric

$$PP(W) = P(s_1, s_2, \ldots, s_m)^{-1/m}$$

- W: test set containing m sentences s
- $s_1, s_2, \ldots, s_m$: sentence ending with </s>
- m: # all words in W including </s> but not <s>

→ measure complexity of a sample of texts

- Smaller perplexity = better model

  good model: PP(W): 60 → 20 (or lower)

- Character level models PP < word-based models PP

- Perplexity for bigram models

$$PP(W) = \sqrt[m]{\prod_{i=1}^{m} \prod_{j=1}^{|s_i|} \frac{1}{P(w_j^{(i)} \mid w_{j-1}^{(i)})}}$$

$w_j^{(i)}$ → jth word in ith sentence

→ concatenate all sentences in W

$$PP(W) = \sqrt[m]{\prod_{i=1}^{m} \frac{1}{P(w_i \mid w_{i-1})}}$$

w_i → ith word in test set

$$\log PP(W) = \frac{-1}{m} \sum_{i=1}^{m} \log_2 \left(P(w_i \mid w_{i-1}) \right)$$

+) Out of vocab words

Speech recog or question answering

- Closed vs. open vocabularies
- Unknown word = out of vocab word (OOV)
- special tag <UNK> in corpus and in input

→ How to create vocab V

- Min word frequency f
- Max |V|, including words by frequency
- Use <UNK> sparingly
- Perplexity - only compare LMs with the same V

+) Smoothing

• Add-one smoothing (Laplacian Smoothing)

$$P(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + V}$$

• Add K-Smoothing

$$\frac{\dots + K}{+K+V}$$

• Advanced smoothing

- Kneser-Ney smoothing
- Good-Turing Smoothing

+) Backoff

• if N-gram missing => Use (N-1)gram...

- Prob discounting e.g Katz backoff
- "Stupid" backoff : $(N-1) * C \quad (C = 0,4)$

e.g PC chocolate | John drinks)

⇓

0,4 x P (chocolate | drinks)

+) Interpolation

$\hat{P}$(chocolate | John drinks) = 0,7 x P(chocolate | John drinks)

+0,2 P(chocolate | drinks) +0,1 x P (chocolate

$\sum_i \lambda_i = 1$ → optimize lambda using validation set

(8) Word Embeddings

a) Applications
- Semantic analogies and similarity
- Sentiment analysis
- Classification of customer feedback
- Machine translation
- Information extraction
- Question answering

→ Continuous bag of words or CBOW
- try to predict the particular word given a few surrounding text words
- the weights used to predict → word vectors

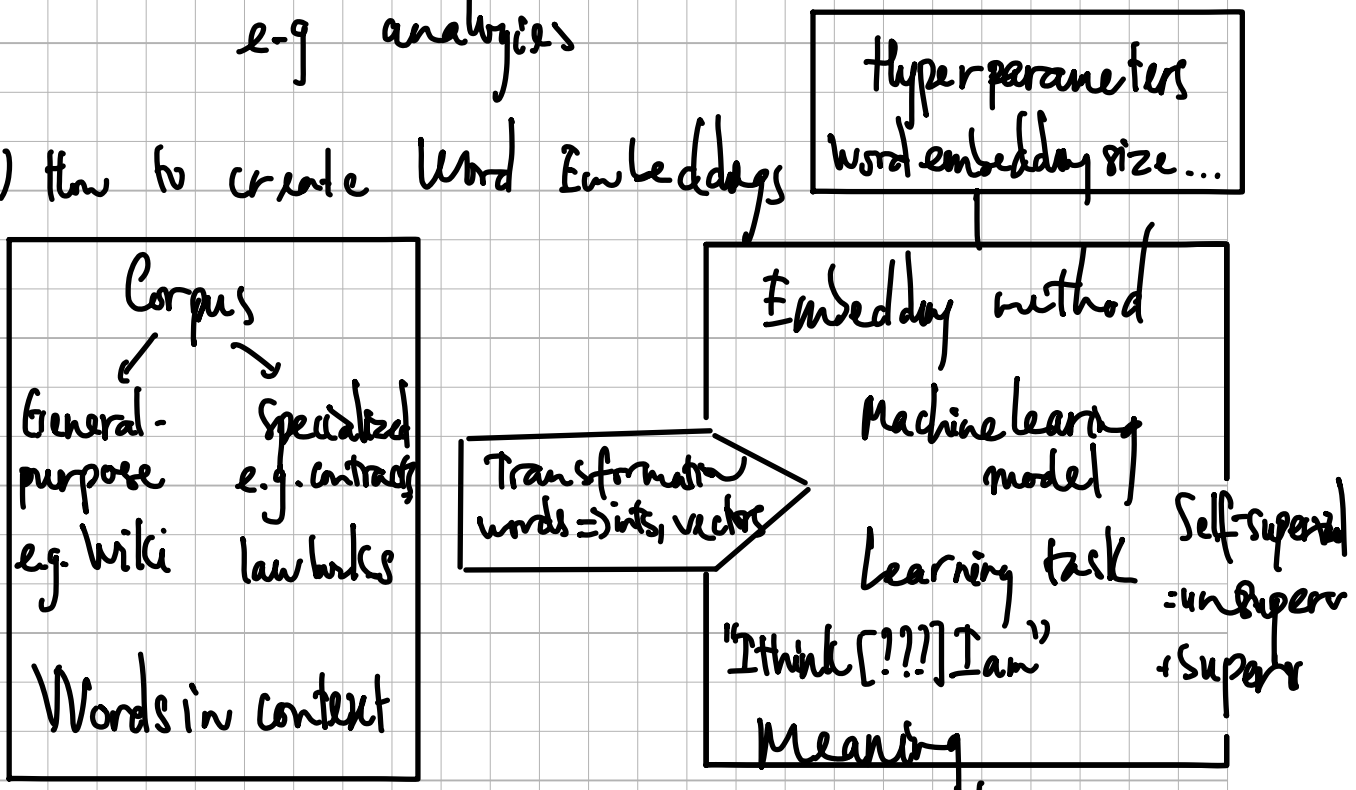
weight for words predict
weight for words content

⇒ manipulate 2 weights

b) Basic Word representations
- Integers → cons: implied ordering not good
- One-hot vectors ⇄ pros: simple, no implied, cons: huge, encode no meaning
- Word embeddings

+) Word embeddings
- low dimension
- Embed meaning
e.g semantic distance
e.g analogies

+) How to create Word Embeddings



"the main product" → Word embeddings

? What will ultimately define the meaning of the individual words? the specific of task
=) The meaning of words, as carried by the word embeddings, depends on the embedding approach

+) Word Embedding methods
- word2vec (Google, 2013)
- Continuous bag-of-words (CBOW)
- Continuous skip-gram / Skip-gram with negative sampling (SGNS)

- Global Vectors (GloVe) (Standford, 2014)

- fastText (Facebook, 2016)
  - Supports out-of-vocabulary (OOV) words

Advanced word embedding methods

→ Deep learning, contextual embeddings

Tunable pre-trained models available
- BERT (Google, 2018)
- ELMo (Allen Institute for AI, 2018)
- GPT-2 (OpenAI, 2018)

+) Continuous Bag-of-words Model

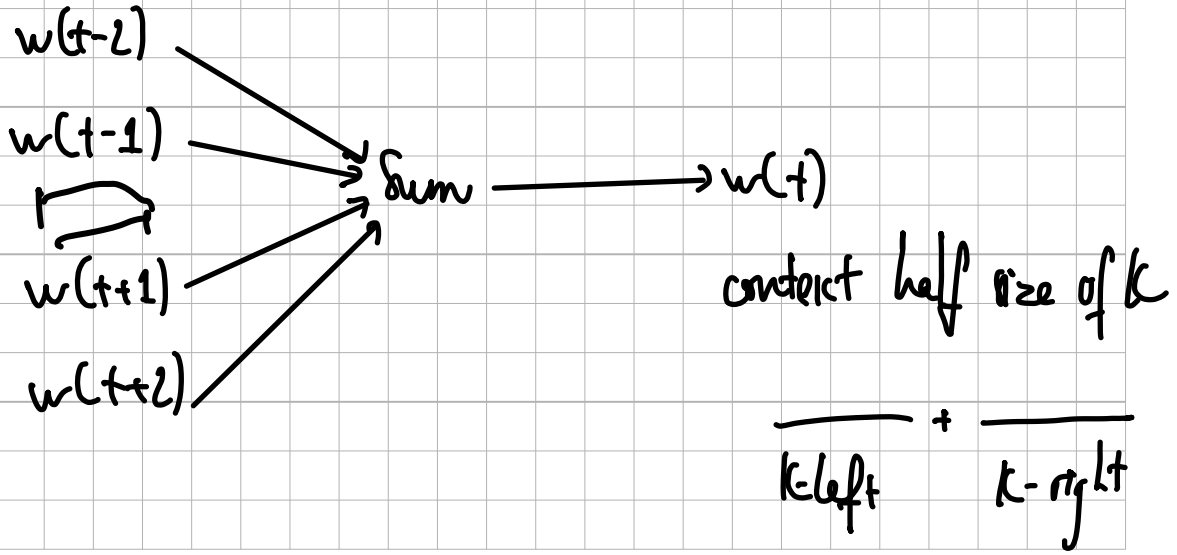
Center word prediction: rationale

e.g. the little ? is barking

↓

dog puppy hound terrier ...

- CBOW in a nut shell



context half size of K

$\frac{}{K\text{-left}} + \frac{}{K\text{-right}}$

objective: to predict a missing word based on surrounding words

+) Cleaning & Tokenization

- Letter case "the" == "the" == "THE" → lowercase/uppercase
- Punctuation , ! . ? → . "'cc,,'" → ϕ ...!!??! → .
- Numbers 1 2 ... → ϕ 3.14159 90210 → as is / <NUMBER>
- Special characters ∇ $ € § ¶ ** → ϕ
- Special words ☺ #nlp → :happy: #nlp

+) Sliding Window of words

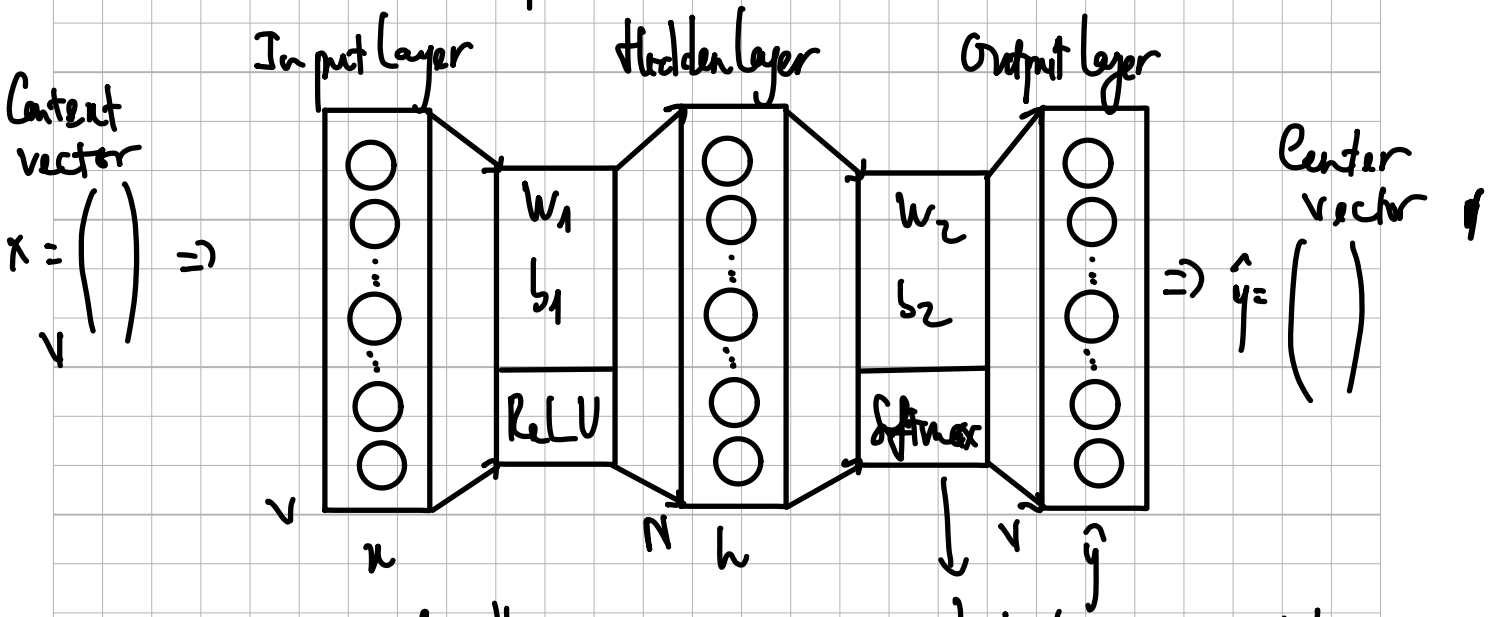
```
i : C                                  list of word
while i ( len(words) - C:  → context size
    center = word[i]
    context = words[(i-C):i] + words[(i+1):(i+C+1)]
    yield context, center  # x, y
    i += 1
```

+) Transform words into vectors

- Started with one-hot vectors for the context words
→ transform into a single one : taking average !

Context words	Context vec	Center word	Center vec
I am because I	[0,25; 0,25; 0,5; 0]	happy	[0,0; 1,0; 0]

+) Architecture of the CBOW model



X: average of all context vectors distribution over V

pick output = argmax of output

+) Dimensions

x ($V \times 1$) $\rightarrow$ W_1 $N \times V$, b_1 $N \times 1$ $\rightarrow$ h ($N \times 1$) $\rightarrow$ W_2 $V \times N$, b_2 $V \times 1$ $\rightarrow$ $\hat{y}$ ($V \times 1$)

ReLU

$z_1 = W_1 x + b_1$ $N \times 1$

$h = \text{ReLU}$ $N \times 1$

Softmax

$z_2 = W_2 h + b_2$ $V \times 1$

$\hat{y} = \text{softmax}(z_2)$ $V \times 1$

- np.mean(context_words_vectors, axis=0)
- Batch processing

$\hat{Y} = \left(\left(\hat{y}^{(1)} \right) \cdots \left(\hat{y}^{(m)} \right) \right)$

Context words vectors

$X = \left(\left(x^{(1)} \right) \cdots \left(x^{(m)} \right) \right)$ V ↕, m → batch size

$X : V \times m$

$W_1 : N \times V$ $W_2 : V \times N$

$b_1 : N \times m$ $b_2 : V \times m$

$H : N \times m$ $Z_2 : V \times m$

$Z_1 : N \times m$ $\hat{Y} : V \times m$

V = 8000 batches of 16

m = 16 W_1: N x V b_1, b_2 broad cast

N = 400 ↓ N x 1 ↓ V x 1

+) Rectified linear unit (ReLU)

ReLU(x) = max(0, x).

Soft max

z (V) $\begin{pmatrix} z_1 \\ \vdots \\ z_V \end{pmatrix}$ $\xrightarrow{\text{Softmax}}$ $\begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_V \end{pmatrix}$ (V) Probabilities of being center word

$$\hat{y}_i = \frac{e^{z_i}}{\sum_{i=1}^{V} e^{z_i}}$$

+) Training a CBOW model

Cost function

↳ cross-entrophy loss → classification model

$$J = -\sum_{k=1}^{V} y_k \log \hat{y}_k = \log a$$

wrong: $J = -\log \hat{y}_k$

incorrect predictions: penalty

correct predictions: reward

+) forward propagation

Loss → single example

Cost → a batch of examples = mean of losses.

$$J_{batch} = \frac{-1}{m} \sum_{i=1}^{m} \sum_{j=1}^{K} y_j^{(i)} \log \hat{y}_j^{(i)} = \frac{1}{m} \sum_{i=1}^{m} J^{(i)}$$

+) Backpropagation and Gradient Descent

$$\frac{\partial J_{batch}}{\partial W_1} = \frac{1}{m} \left(W_2^T (\hat{Y} - Y) \cdot step(Z_1)\right) X^T$$

$$\downarrow \quad ReLU(W_2^T (\hat{y} - y)) x^T$$

$$\frac{\partial J_{batch}}{\partial W_2} = \frac{1}{m} (\hat{Y} - Y) H^T$$

$$ReLU(W_2^T (\hat{y} - y))$$

$$\frac{\partial J_{batch}}{\partial b_1} = \frac{1}{m} \left(W_2^T (\hat{Y} - Y) \cdot step(Z_1)\right) \overbrace{1_m^T}^{\top}$$

↳ np.sum(a, axis=1, keepdims=True)

$$\frac{\partial J_{batch}}{\partial b_2} = \frac{1}{m} (\hat{Y} - Y) 1_m^T$$

Minimize: $W := W - \alpha \, dW$

$b := b - \alpha \, db$

+) Extracting Word Embedding Vectors

option 1 : W_1 : $N \times V$ $\quad x = (V, 1)$

→ each col = w each word

Option 2 : W_2 : $V \times N$ $\quad x = (V, 2)$

→ each row = w each word

option 3

$W_3 = 0,5 (W_1 + W_2^T) = N \times V$

→ each col : w each word

+) Evaluating Word Embeddings

Intrinsic Evaluation

Continuous Skip grams model vs. Continuous bag-of-words model

↳ predict contexts given center | predict target given surroundings

allows you to test relationships between words

↳ Semantic analogies, syntactic analogies.

? harder to track: Ambiguous cases

→ • Analogies
• Clustering
• Visualization

→ Word embeddings were created by Continuous Skip-grams, not Contd bag-of-words

Extrinsic Evaluation

Test word embeddings on external task

e.g named entity recognition, parts-of-speech tagging

+ Evaluates actual usefulness of embeddings

- Time - consuming
- More difficult to trouble shoot

library: Keras, Pytorch

(9) Recurrent Neural Networks for Language Modeling

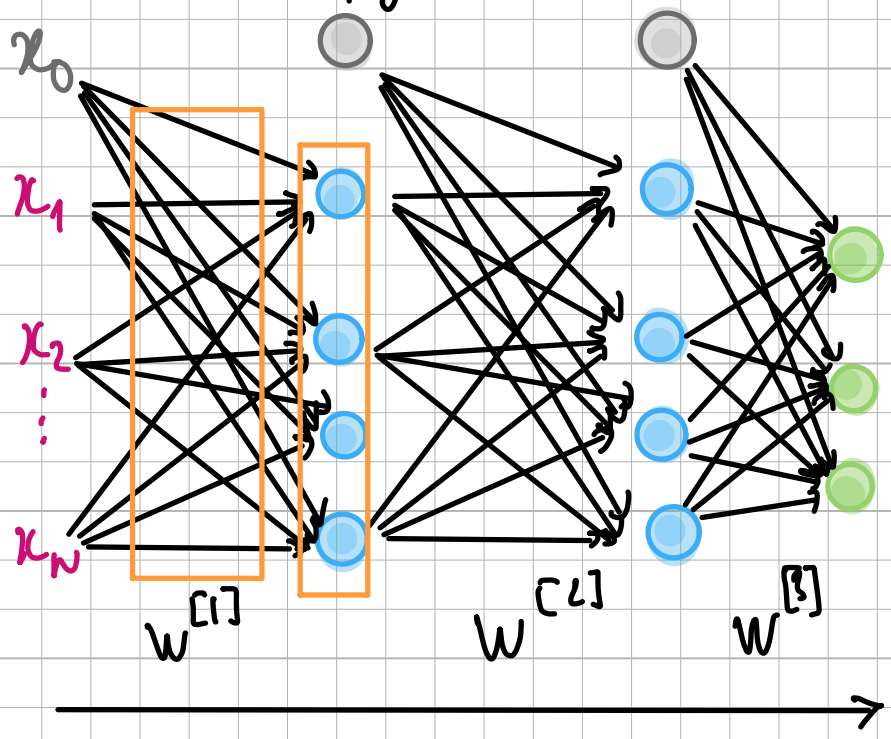
1) Introduction

Deep neural networks can learn more abstract information and do not require any feature engineering.

2) Neural Networks for Sentiment Analysis

- Neural networks and forward propagation

Forward propagation



$a^{[i]}$ Activations ith layer

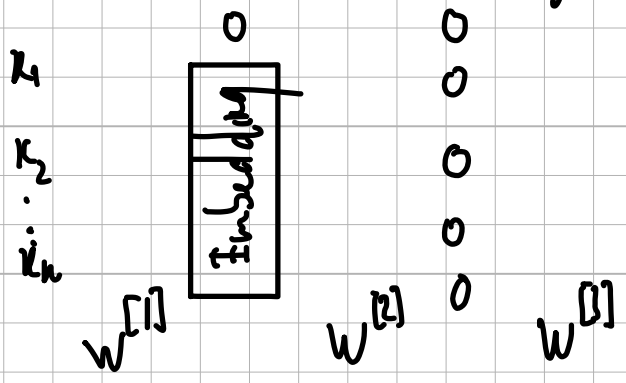
$a^{[0]} = X$

$z^{[1]} = W^{[1]} \cdot a^{[i-1]}$

$a^{[i]} = g^{[i]}(z^{[i]})$

↑ activation func

- Sentiment Analysis



0 - -> Positive (1)

0 - -> Negative (0)

+) Initial Implementation
+ padding

+) Dense Layers and ReLU

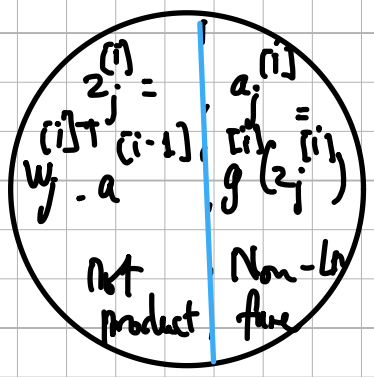
→ Using tensorflow

tf.keras.layers.Dense(32)

tf.keras.layers.ReLU()

= tf.keras.layers.Dense(32, activation='relu')

→ Dense layer with 32 units, ReLU layer



Dense layer

$z^{[i]} = W^{[i]} a^{[i-1]}$

↓

trainable params

ReLU layer

$g(z^{[i]}) = \max(0, z^{[i]})$

+) Embedding and Mean layers

- Embedding layer

  Trainable weights : Vocabulary x Embedding

- Mean layer take the average of the embeddings.

  Same size as # N C features)

  Does not have any trainable params

→ N grams vs. Sequence Models

+) traditional language models

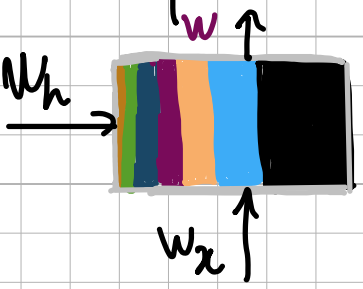
• limitations:

- large N-grams needed to capture dependencies between distant words
- Need a lot of space and RAM

+) Recurrent Neural Networks

• RNNs model relationships among distant words

• In RNNs a lot of computations share parameters



+) Applications of RNNs

One to One : given stores → predict winner

One to Many < inputs a sentence and generates a sentence
inputs a topic and generates a conversation about that topic

Many to one : gives a tweet → predict sentiment

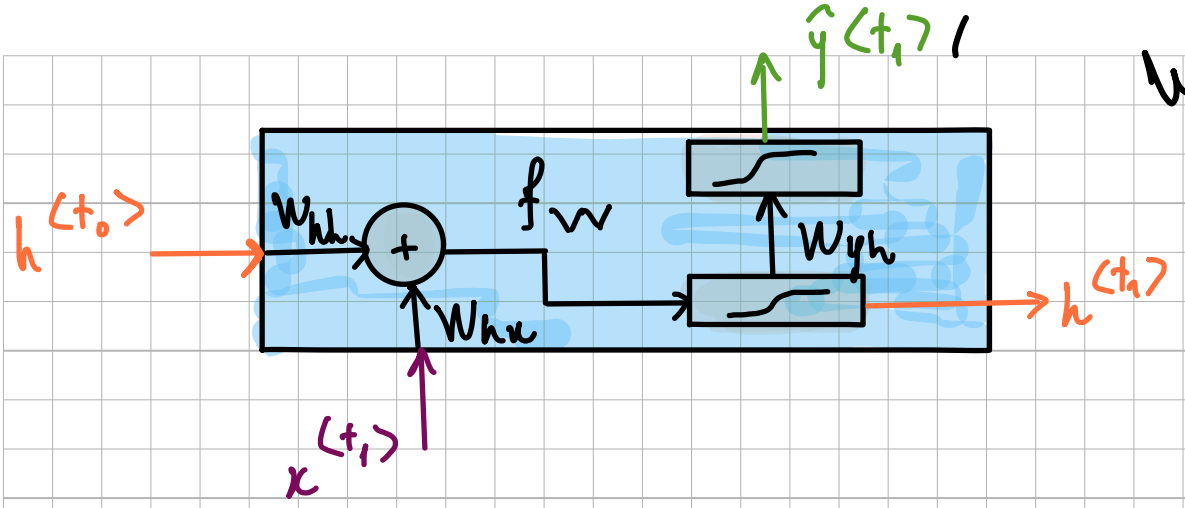
Many to many : machine translation

+) Math in Simple RNNs

• Hidden states propagate information through time

• Basic recurrent units have two inputs at each time

$h^{<t-1>}$, $x^{<t>}$



W → trans

Note that:

$$h^{\langle t \rangle} = g\left(W_h \left[h^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_h\right)$$

is the same as multiplying W_{hh} by h and W_{hx} by x

$$h^{\langle t \rangle} = g\left(W_{hh} h^{\langle t-1 \rangle} + W_{hx} x^{\langle t \rangle} + b_h\right)$$

$$\hat{y}^{\langle t \rangle} = g\left(W_{yh} h^{\langle t \rangle} + b_y\right)$$

→ training W_{hh}, W_{hx}, W_{yh}, b_h, b_y

$[h \times h, h \times x] \rightarrow h \times (h+x)$

$W_h = [W_{hh} \mid W_{hx}]$ horizontal concatenation

→ Option 1: np.concatenate((W_hh, W_hx), axis = 1)

→ Option 2: np.hstack((W_hh, W_hx))

$$\left[h^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] = \begin{bmatrix} h^{\langle t-1 \rangle} \\ x^{\langle t \rangle} \end{bmatrix}$$

→ Option 1: np.concatenate((h_tprev, x_t))

→ Option 2: np.vstack((h_tprev, x_t))

- Formula 1:

np.matmul(np.hstack((w_hh, w_hx)), np.vstack((h_t-prev, x_t)))

- Formula 2:

np.matmul(w_hh, h_t-prev) + np.matmul(w_hx, x_t)

→ np.allclose(formula_1, formula_2) : True

+) Cost function for RNNs

for RNNs, the loss function is just an average through the

$$J = \frac{-1}{T} \sum_{t=1}^{T} \sum_{j=1}^{k} y_j^{(t)} \log \hat{y}_j^{(t)}$$

+) Implementation note:

scan() function in tensorflow (GPU, compute in parallel)

scan(fn, elems, initializer=None, ...)

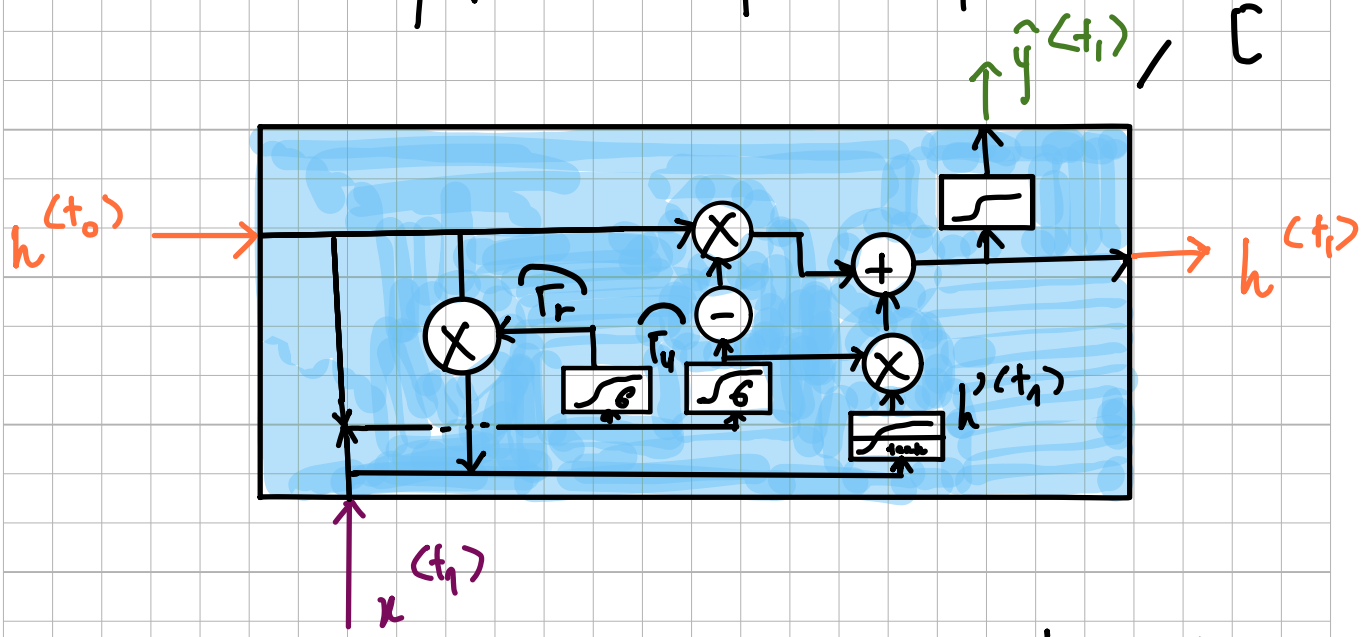
↓ fw ↓ $x^{(1)} \dots x^{(T)}$ ↓ $h^{(t_0)}$

return ys, cat value

+) Gated Recurrent Units (GRU)

Relevance and update gates to remember important prior information

- GRUs "decide" how to update the hidden state
- GRUs help preserve important information



$\Gamma_u = \sigma(W_u [h^{\langle t_0 \rangle}, x^{\langle t_1 \rangle}] + b_u]$ → update gate

$\Gamma_r = \sigma[W_r [h^{\langle t_0 \rangle}, x^{\langle t_1 \rangle}] + b_r]$ → find relevance score

$h'^{\langle t_1 \rangle} = \tanh(W_h [\Gamma_r * h^{\langle t_0 \rangle}, x^{\langle t_1 \rangle}] + b_h)$

$h^{\langle t_1 \rangle} = (1 - \Gamma_u) * h^{\langle t_0 \rangle} + \Gamma_u * h'^{\langle t_1 \rangle}$

$\hat{y}^{\langle t_1 \rangle} = g(W_y h^{\langle t_1 \rangle} + b_y)$

$c = np.tanh(c)$ return_sequences = True

batch_size

sequence_length → not fixed, "None", expect any

word_vector_length → fixed in model

model.GRU.build([None, None, word_vector_length])
.summary()

+) Deep and Bi-directional RNNs

- BiRNN
  acyclic graph (không chu trình)
  ↳ Information flows from the past and from the future independently

$$\hat{y}^{\langle t \rangle} = g(W_y[\overrightarrow{h}^{\langle t \rangle}, \overleftarrow{h}^{\langle t \rangle}] + b_y)$$

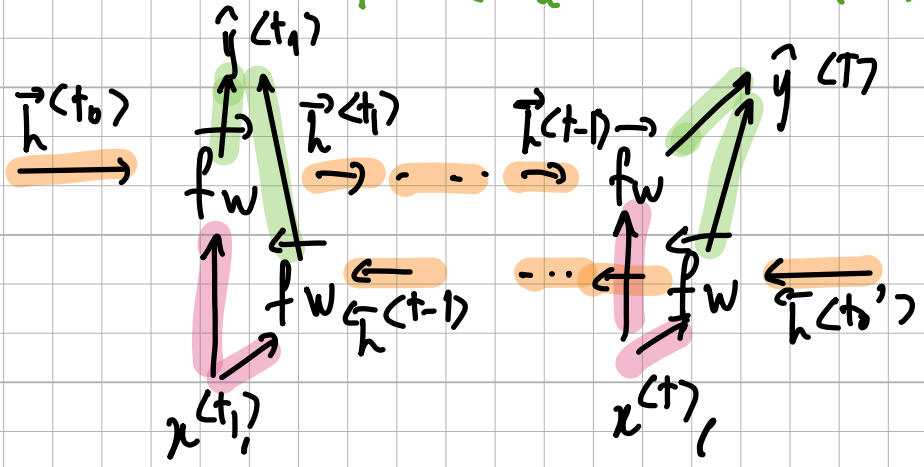
- Deep RNN

1. Get hidden states for current layer →

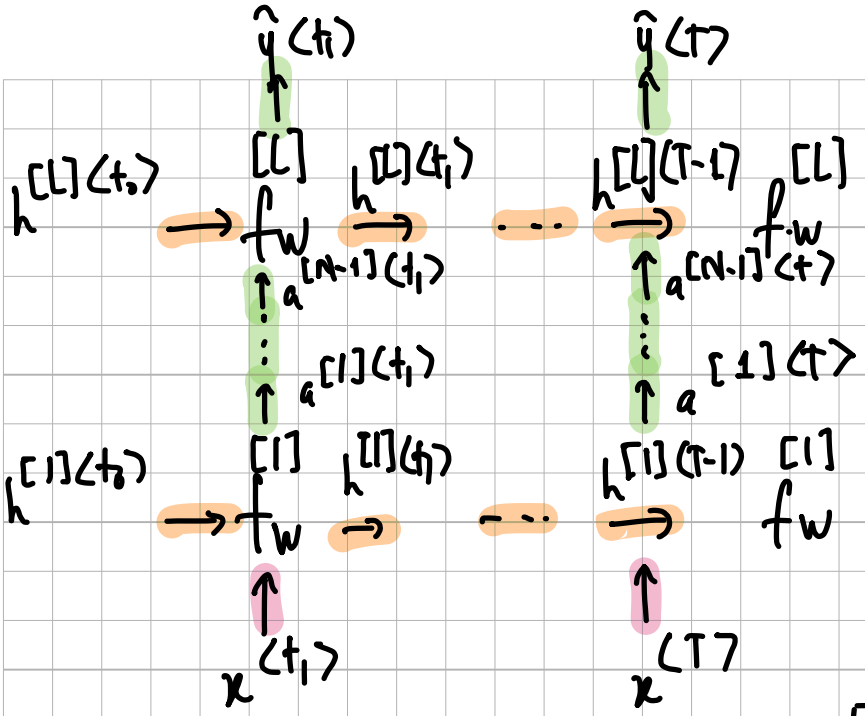
$$h^{[l]\langle t \rangle} = f^{[l]}(W_h^{[l]}[h^{[l]\langle t-1 \rangle}, a^{[l-1]\langle t \rangle}] + b_h^{[l]})$$

2. Pass the activations to the next layer ↑

$$a^{[l]\langle t \rangle} = f^{[l]}(W_a^{[l]} h^{[l]\langle t \rangle} + b_a^{[l]})$$



Bi RNNs.



Deep RNN

Note: At layer h_i, using $a^{[l-1]}$, $h^{[l]}$

+) Perplexity

$$PP(W) = \sqrt[N]{\prod_{i=1} \frac{1}{P(w_i \mid w_1 \dots w_{i-1})}}$$

$$\log P(W) = \frac{-1}{N} \sum_{i=1}^{N} \log P(w_i \mid w_1 \dots w_{i-1})$$

(10) LSTMs and Named Entity Recognition

+) RNNs and Vanishing Gradients

Solving for vanishing or exploding gradients

- Identity RNN with ReLU activation

$\begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$ $-1 \to 0$

- Gradient clipping $32 \to 25$

- Skip connections

explode or vanish

$x \to [\,] \to \oplus \to F(x)+x$

$F(x)+x$

$\sigma \in [0,1]$

$\tanh \in [-1,1]$

$\to$ depends on eigen values λ

Max W $\begin{cases} < \lambda \to \text{vanishing} \\ > \lambda \to \text{explodes} \end{cases}$

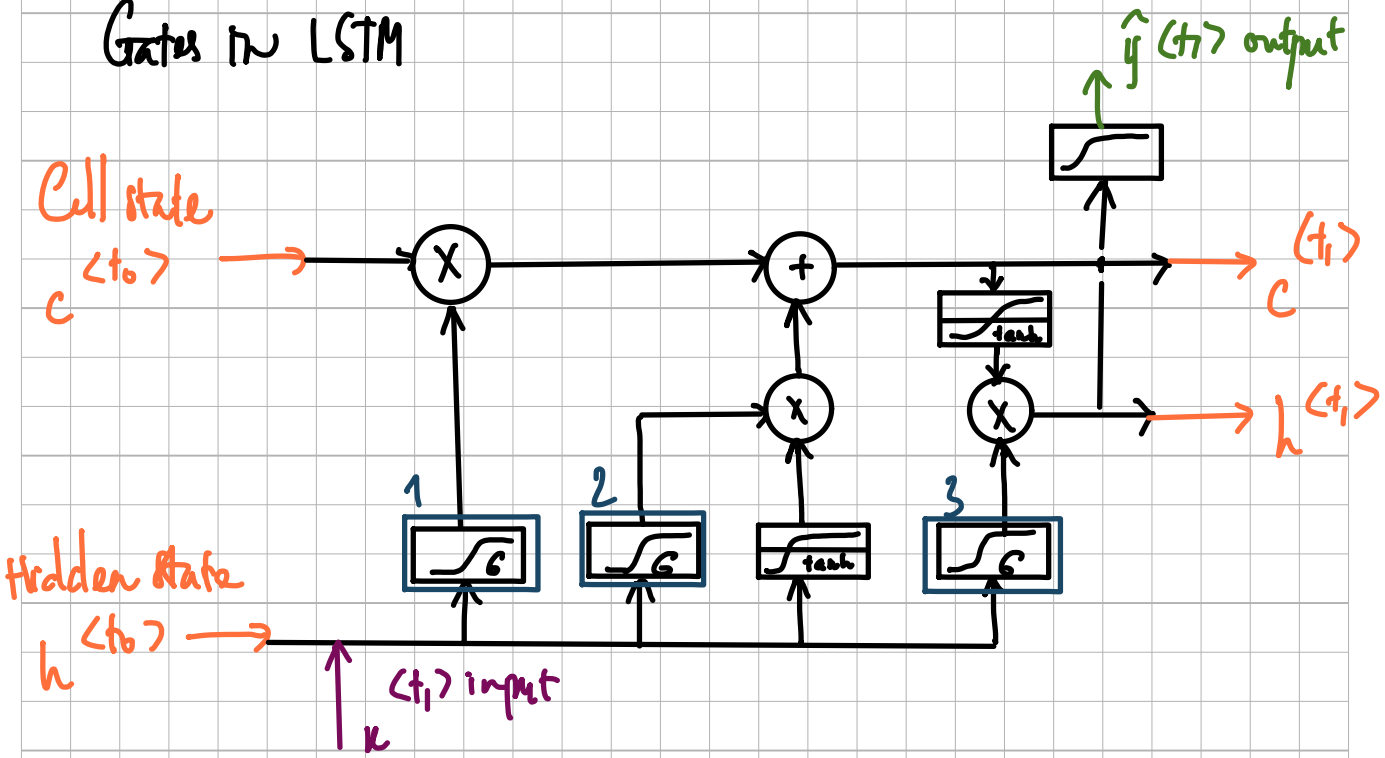
Derivative of activation func bounded by $\alpha = \frac{1}{\lambda}$

$d(\text{sigmoid}) = (\text{sigmoid})(1-\text{sigmoid})$

$\in [0, 0.25]$

+) Intro to LSTMs (Long Short-term memory unit)

Gates in LSTM



1. Forget gate (to keep) f
   information that is no longer important
2. Input gate (to add) i
   information to be stored
3. Output gate (next hidden states will be) o
   information to use at current step

Applications

- Next-character prediction (email?)
- Chatbots
- Music composition
- Image Captioning
- Speech recognition

+) LSTM architecture

The gate gate (candidate memory cell):

$$g = \tanh(W_g [h_{t-1}; x_t] + b_g)$$

The cell state:

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

The output of LSTM unit

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

+) Intro to NER

locates and extracts predefined entities from text

Places, organizations, names, time and dates

Types of entities

Geographical	Time indicator
organization	Artifact
Geopolitical	Person

Applications

Seach efficiency

Recommendation engines

Customer service

Automatic trading (news...)

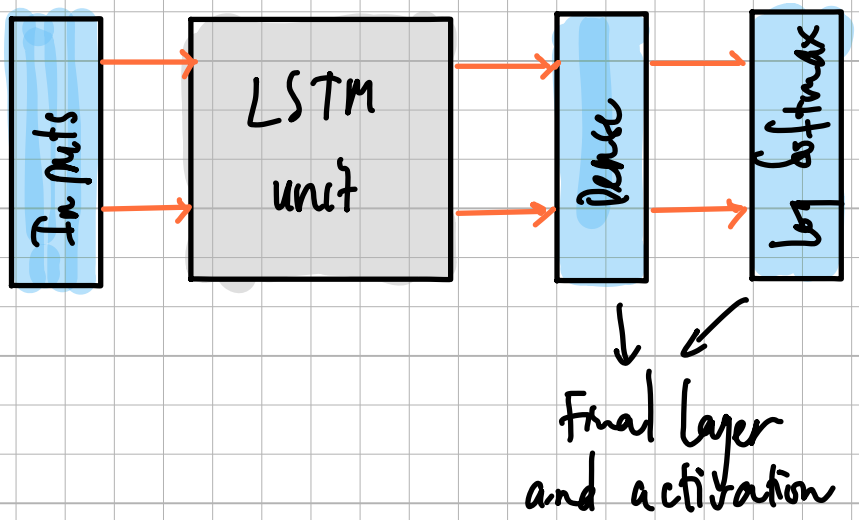
f) Training NERs

Data Processing

- Convert words and entity classes into arrays (same-length)
- Pad with tokens: Set sequence length to a certain number and use <PAD> token to fill empty spaces
- Create a data generator

Training an NER system

1. Create a tensor for each input and its corresponding number
2. Put them in a batch → 64, 128, ... (faster processing)
3. Feed it into an LSTM unit
4. Run the output through a dense layer
5. Predict a log softmax over K classes



f) Computing accuracy

- Pass test set through the model

- Get arg max across the prediction array
- Mask padded tokens
- Compare with true labels

(11) Siamese networks and the triplet loss

Identify similarity between things

similarity

+) Applications:

Handwritten checks

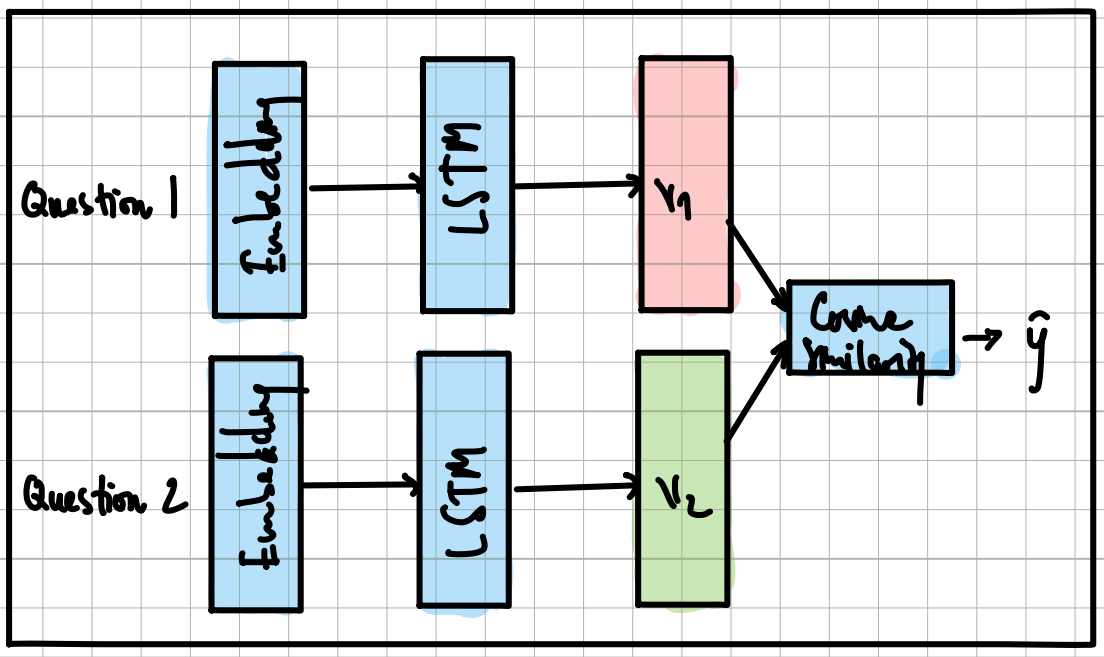
Question duplicates / Quora

Queries

+) Architecture

Subnetworks share identical parameters, only needs train one

1. Embedding
2. LSTM Coptimal
3. Vectors (output of each sub-network)
4. Cosine Similarity



lab

→ Creating a Siamese model

```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Model, Sequential
from tensorflow import math
import numpy
# Setting random seeds
numpy.random.seed(10)
```

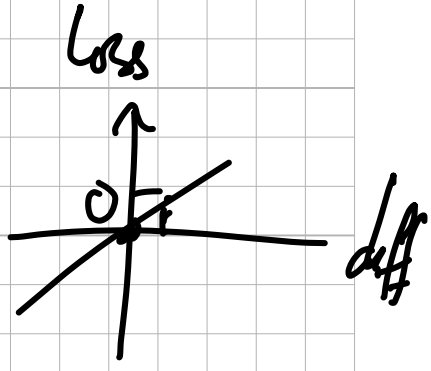
+) Cost function : Triplet loss

Anchor, Positive, Negative

$s(A, P) \approx 1$

$s(A, P) \approx -1$

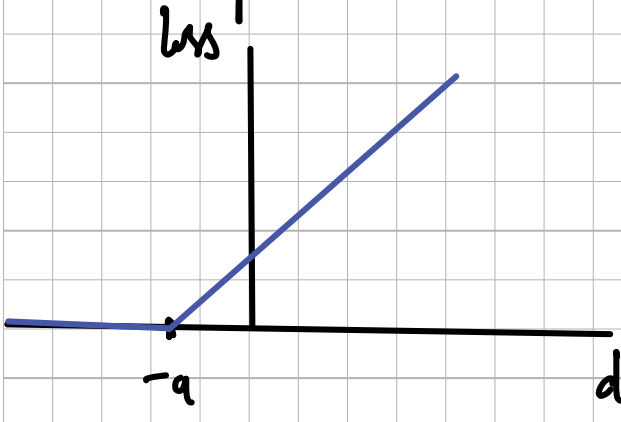
$diff : s(A, N) - s(A, P)$



$-\cos(A, P) + \cos(A, N) \leq 0$

prediction	$-\cos(A, P)$	$\cos(A, N)$	sum
☑	$-(1)$	-1	-2
☒	$-(-1)$	1	2

+) Triplets loss



with alpha margin

$$\mathcal{L} = \begin{cases} 0; & \text{if } diff + \alpha \leq 0 \\ diff + \alpha; & \text{if } diff + \alpha > 0 \end{cases}$$

Full cost: $\max(-\cos(A,P) + \cos(A,N) + \alpha, 0)$

↓ controls how far $\cos(A,P)$ is from $\cos(A,N)$

- easy neg triplet:

  $\cos(A,N) < \cos(A,P)$

- Semi-hard neg triplet:

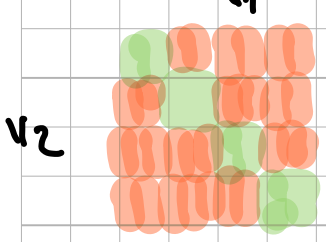
  $\cos(A,N) < \cos(A,P) < \cos(A,N) + \alpha$

- Hard neg triplet:

  $\cos(A,P) < \cos(A,N)$

+) Computing the cost:

$s(v_1, v_2)$



$\mathcal{L}(A,P,N) = \max(diff + \alpha, 0)$

$diff = s(A,N) - s(A,P)$

$$J = \sum_{i=1}^{m} \mathcal{L}(A^{(i)}, P^{(i)}, N^{(i)})$$

→ the diag line corresponds to scores of similar sentences, (normally they should be positive). The off-diagonals correspond to cos scores between the anchor and the neg examples

2 concepts:

- mean_neg: speeds up training
- closest_neg: helps penalize the cost more

$$\mathcal{L}_{original} = \max(-\cos(A,P) + \cos(A,N) + \alpha, 0)$$

$$\mathcal{L}_1 = \max(-\cos(A,P) + mean\_neg + \alpha, 0)$$

$$\mathcal{L}_2 = \max(-\cos(A,P) + closest\_neg + \alpha, 0)$$

$$\mathcal{L}_{full} = \mathcal{L}_1 + \mathcal{L}_2$$

+) One shot learning

Classification vs. One shot learning

classifies input as 1 of K classes

measures similarity between 2 classes

→ no need to retrain, learn a similarity score

$\cos(sig1, sig2) > \tau$ ✓

$\cos(sig1, sig2) \leq \tau$ ✗

+) Training / Testing

Prepare Batches → embedding vector size

$v_1 = (1, d\_model)$

Training

1. Embedding
2. LSTM
3. Vectors
4. Cosine Similarity

Testing

1. Convert two inputs into a array of numbers
2. Feed it into your model
3. Compare v_1, v_2 using cosine similarity
4. test against a threshold T

(12) Neural Machine translation

Adding an attention mechanism to allow the decoder to access all relevant parts of the input sentence regardless of its length

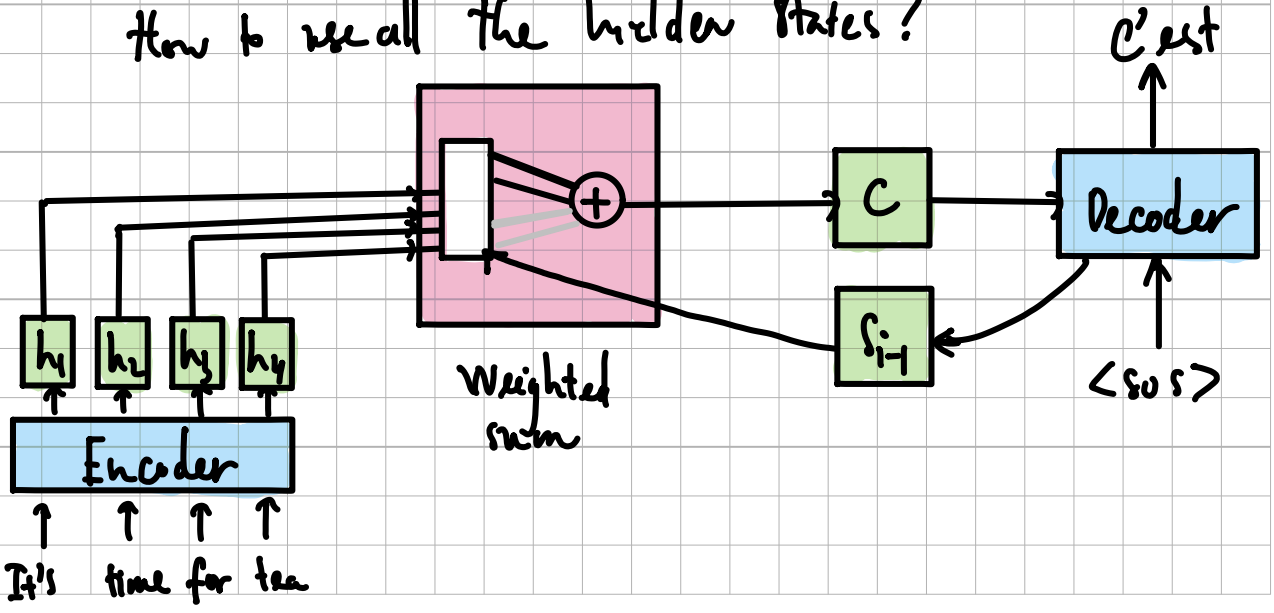
+) Seq2Seq 2014 Google

- Maps variable-length sequences to fixed-length memory
- Inputs and outputs can have different lengths
- LSTMs and GRU to avoid vanishing and exploding gradient problems

The information bottleneck : struggles to compress longer sequences → New layer : Attention

+) Seq2Seq Model with attention

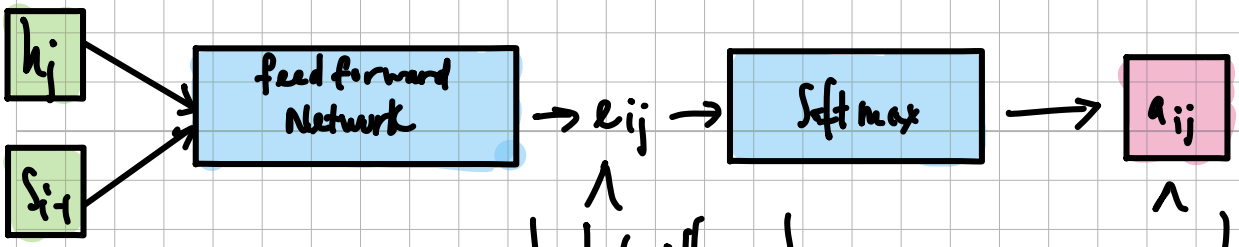
How to recall the hidden states?



the attention layer in more depth

$e_{ij} = a(s_{i-1}, h_j)$ $\quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$

learnable params



calculate the alignment, score of how well the inputs around j match the expected output i

weight used for sum of hidden states

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Context Vector is an expected value

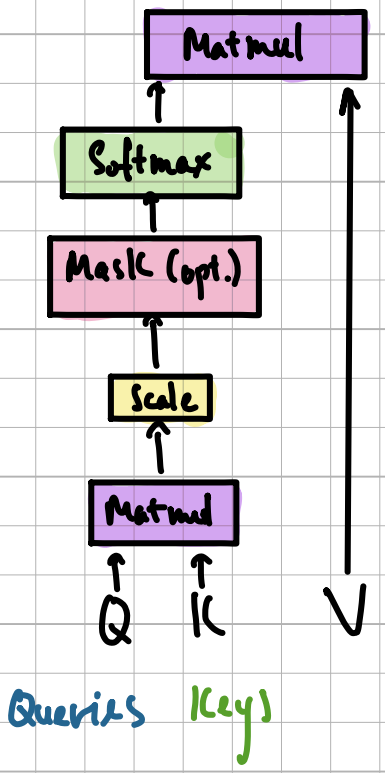
$a_{i1} \cdot h_1 + a_{i2} \cdot h_2 + \dots + a_{im} \cdot h_m \rightarrow c_i$

$e_{ij} = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$

t) Queries, Keys, Values, and Attention

The similarity between words is called alignment of how well query and keys match

Scaled dot-product attention



weight for weighted sum

attention $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

Scale using the root of the key size

→ alignment is learnt in the input embeddings other linear before attention layer

Flexible attention

work for languages with different grammar structures!
despite different ordering

+) Set up for machine translation

- Use pretrained vector embeddings
- otherwise, initially represent words with one-hot vectors
- Keep track of index mappings with word2ind and ind2word dictionaries
- Add end of sequence tokens: <EOS>
- Pad the token vectors with zeros

+) Teacher forcing

Errors from early steps propagate

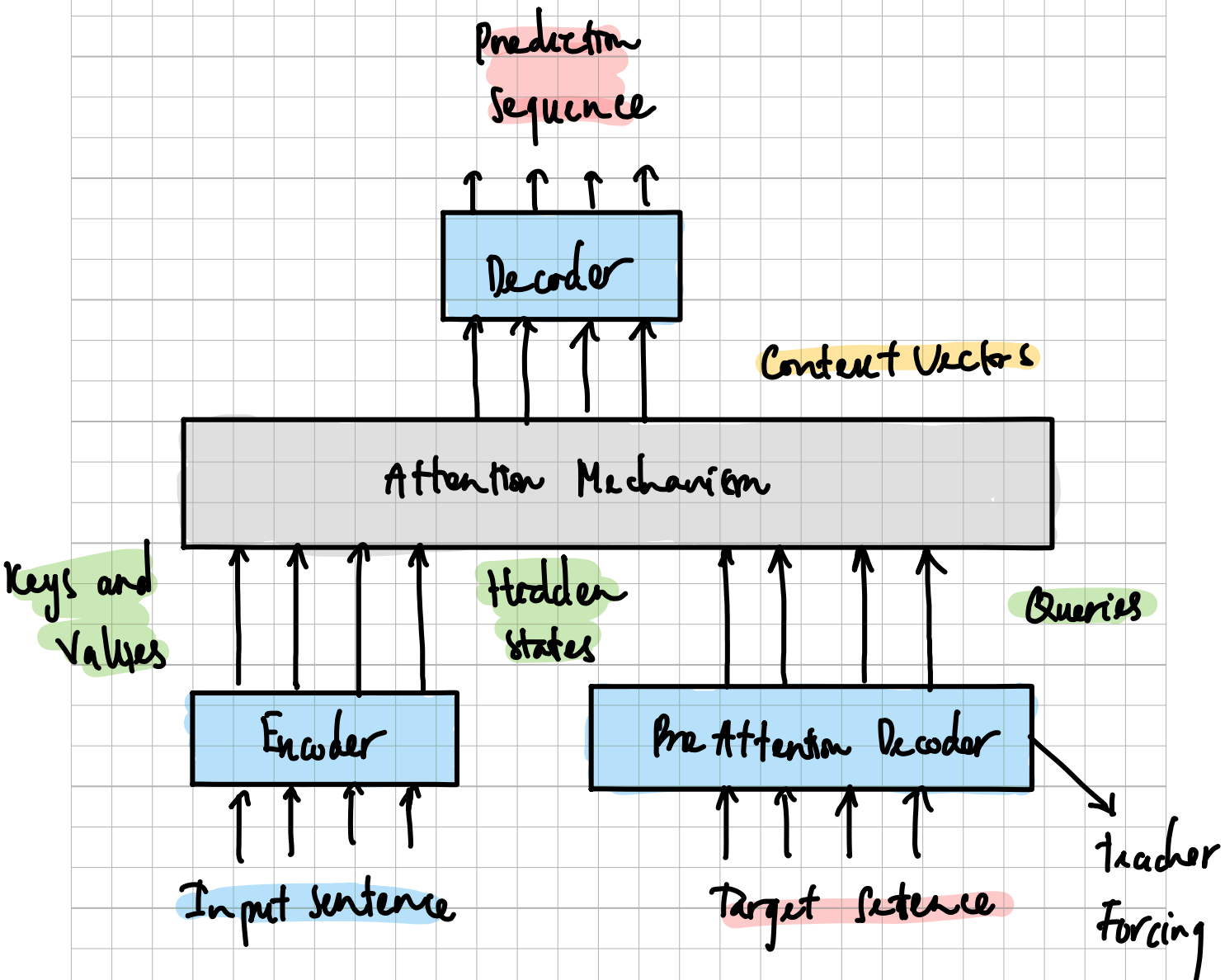
(Naive model)

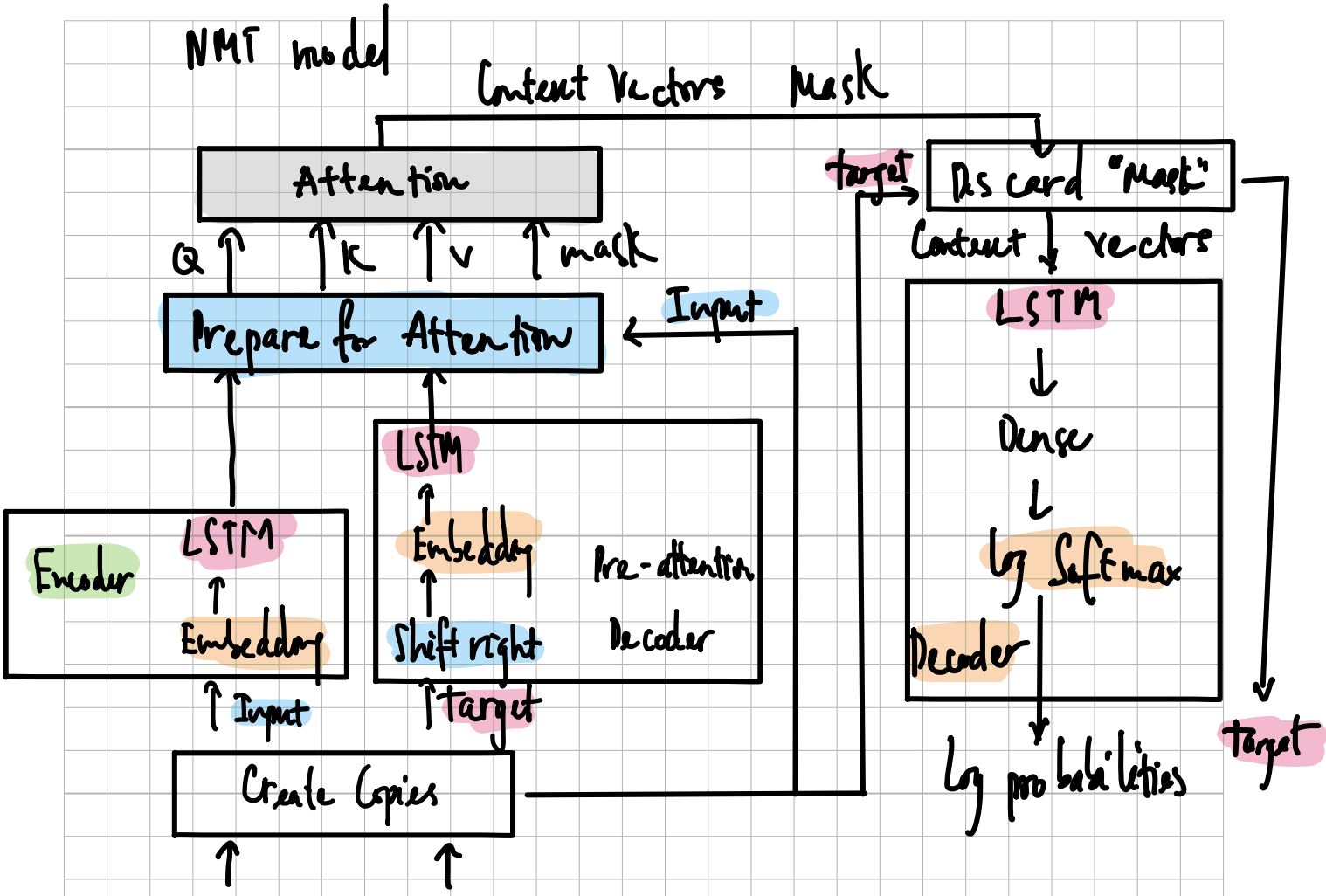
Irl → Correct sequence of words as input (shifted right)

Improves training performance

slowly using decoder outputs

+) NMT with Attention





+) BLEU score (Modified)

Bilingual Evaluation Understudy

Compare candidate translation to reference (human) translations

the better

0 ←——→ 1

Doesn't consider Semantic meaning, Sentence structure

precision

+) ROUGE-N score : recall-oriented

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

$$\rightarrow F_1 = 2 \times \frac{\text{BLEU} \times \text{ROUGE-N}}{\text{BLEU} + \text{ROUGE-N}}$$

only account N-grams

Random Greedy B:1 matching

+) Sampling and Decoding

Temperature control more or less randomness in predictions

lower temperature, more confident, conservative network

higher temperature, more excited, random network

+) Beam search

best sentences over fixed-window size

beam width

→ Calculate probability of multiple possible sequences at each step

Beam width B: # sequences you keep

Until all B most probable sequences end with <EOS>

Beam search decoding

Penalizes long sequences, so you should normalize by the sentence length

Computationally expensive and consumes a lot of memory

+) Minimum Bayes Risk (MBR)

$$\underset{E}{\operatorname{argmax}} \frac{1}{n} \sum_{E'} \mathrm{ROUGE}(E, E')$$

- E: Find the candidate that maximizes
- $\sum_{E'}$: compare with every other candidate
- $\mathrm{ROUGE}(E, E')$: Rouge score between pair of candidates

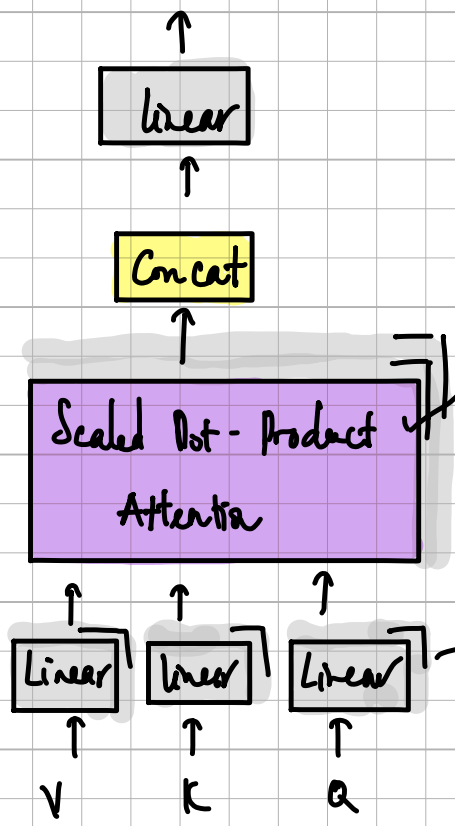
(13) Text summarization

+) Transformers vs. RNNs (no parallel)

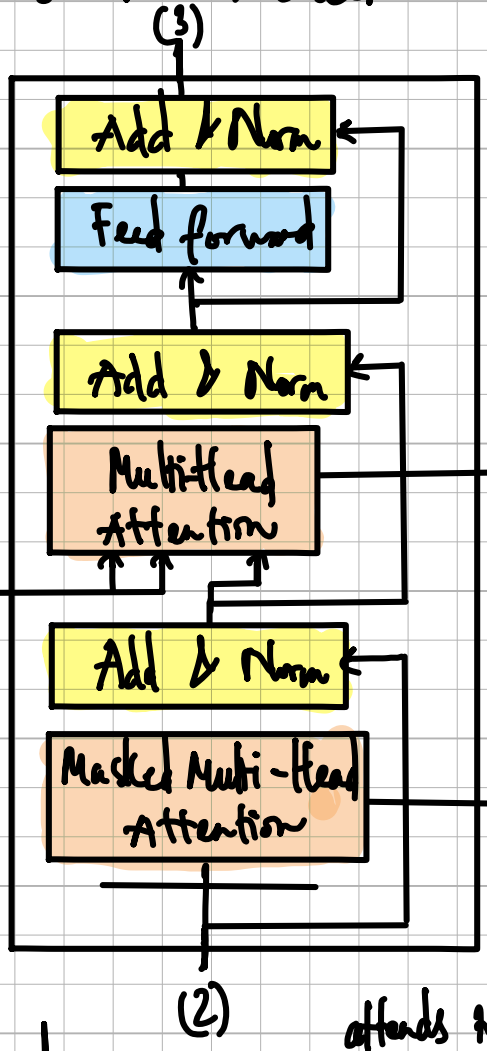
2017 Google

↳ sequential computation, loss of information, vanishing gradients

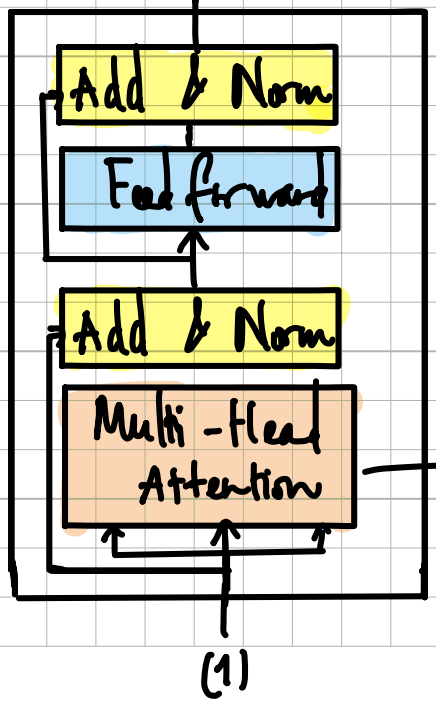
- Multi-Head Attention



- The Decoder



- The Encoder



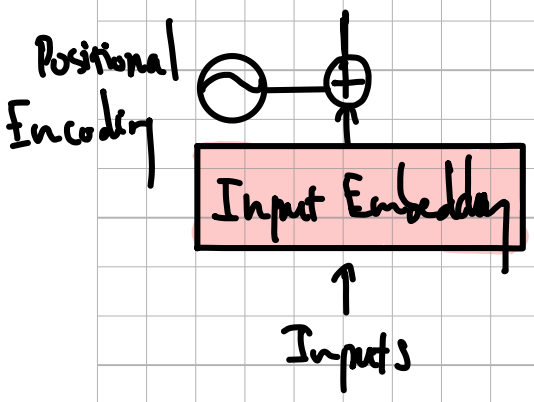
Provides contextual representation of each item in the input sequence

Self Attention

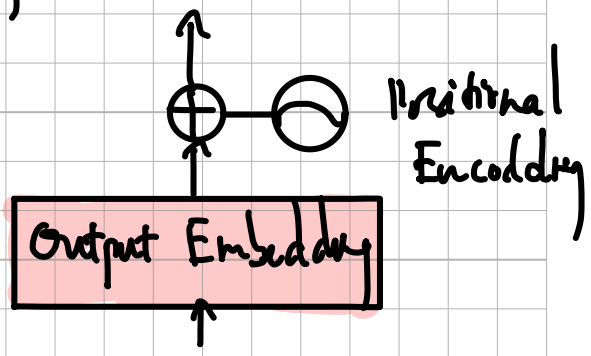
Every item in the input attends to every other item in the sequence

- positional Encoding

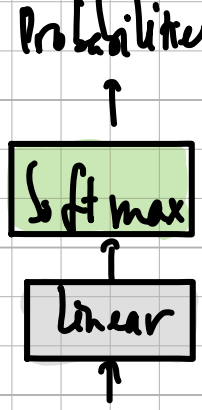
(1)



(2)



(3)



+) Transformer applications

Text summarization — Translation

Autocomplete — Chatbots

NER — Other NLP tasks

Q & A

OpenAI	2018	GPT2
Google AI language	2018	BERT
Google	2019	T_5

T_5 : Text-to-Text transfer Transformer (multitask)

Translation

Classification (Cola sentence)

Q & A

Regression (Stsb)

Summarization

Bi directional Encoder Representations from transformers

Generative Pretraining transformer

+) Scaled and Dot-Product Attention (Heart and Soul)

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V = \text{context vectors for each query } (\#\text{queries} \times \text{size of value vector})$$

w[2,3] weight assigned to third key, second query

Q: stack of rows embedding (generally the same) — n × size of embedding

K, V: same number of rows × size of embedding

Attention Math

+) Masked Self Attention

- Encoder-Decoder Attention

Queries from one sentence,

Keys and values from another

- Self-Attention

Q, K, V comes from the same sentence

Meaning of each word within the sentence

- Masked Self Attention

Q, K, V come from the same sentence

Queries don't attend to future positions

$$\text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}} + \begin{bmatrix} 0 & \cdots & -\infty \\ \vdots & \ddots & \\ 0 & \cdots & 0 \end{bmatrix}\right)$$

+) Multi-Head Attention

sets : # heads

d_{model} = embedding size

W_i^Q W_i^K W_i^V, W^O

$(d_{model} \times \underbrace{\#heads}_{\downarrow d_{HC}} / d_V)$

Usual choice of dimensions

$d_K = d_V = d_{model} / h$

Attention $\rightarrow$ d_V $\rightarrow$ Concat $h \times d_V$

$\rightarrow$ Context vectors for each query

d_{model}

Causal Attention $\simeq$ Masked Attention

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

+) Transformer Decoder

feed Forward (ReLU)

+) Transformer for Summarization

Technical details for data processing

loss weight : 0s until the first <EOS> and then 1s on the start of summary

(0 → 0,2 0,8)

$$J = -\frac{1}{n} \sum_{j}^{n} \sum_{i}^{k} y_j^i \log \hat{y}_j^i$$

i: batch elements

j: over summary

Model Input

[Article] <EOS> [Summary] <EOS>

↳ tokenized input

(14) Question Answering

Context-based Closed-book

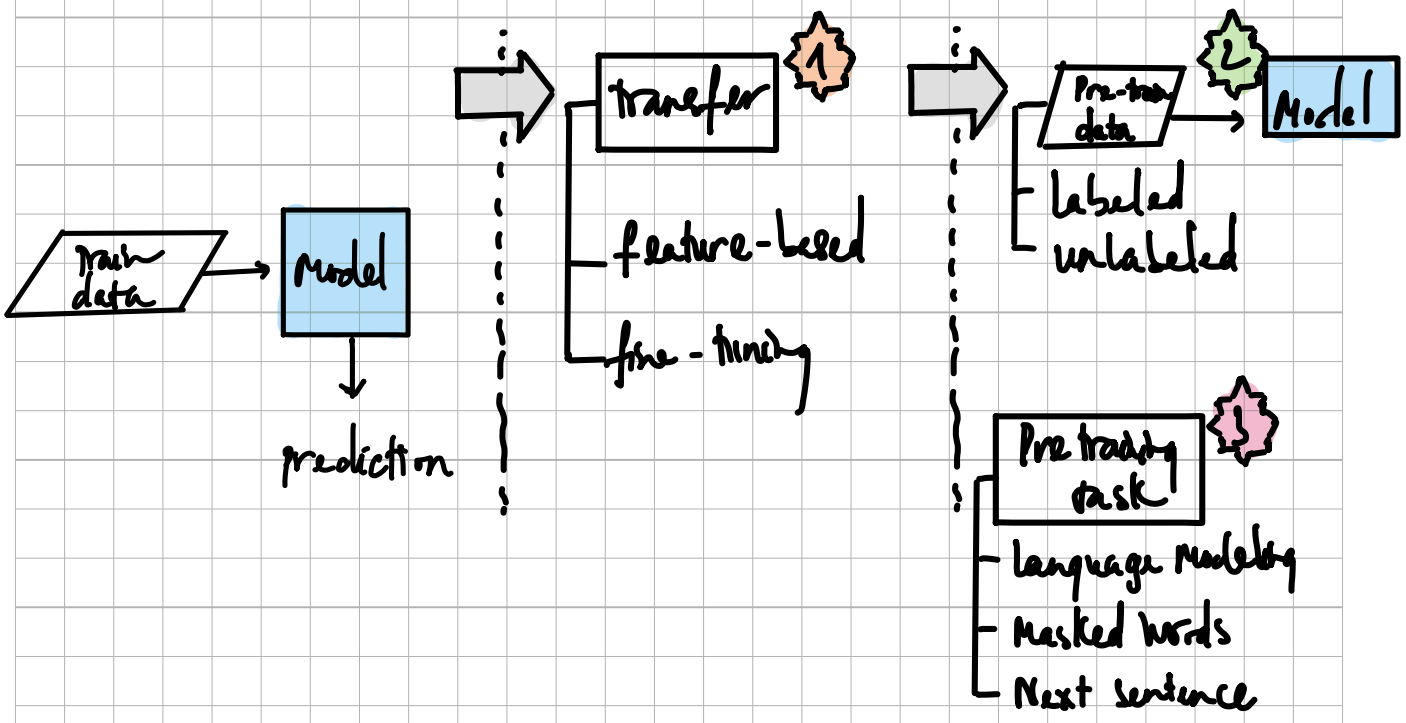
- Transfer learning: reduce training time, improve predictions; small datasets

Pretraining Inference

Training on "Downstream" task

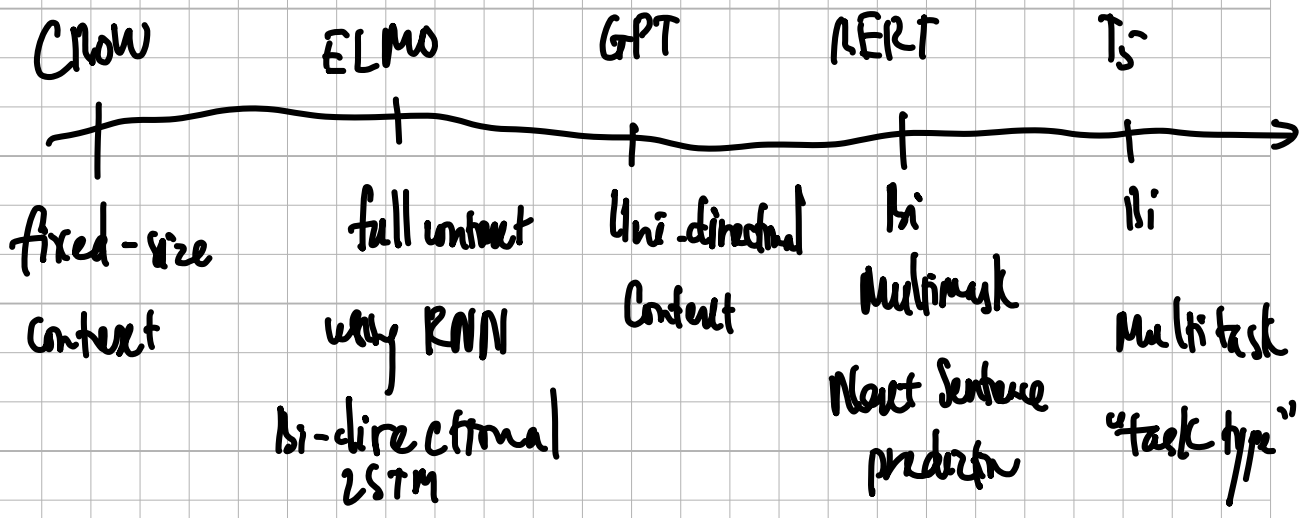
- BERT
- T5: Single task vs. Multi task; more data, better performance

+) transfer learning with full transformer



+) General purpose learning (1)
(2)
Self-supervised tasks (3)

+) ELMO, GPT, BERT, T5



- Transformer
  - Encoder
  - Encoder

GPT
- Decoder

BERT
- Encoder

T5
- Encoder
- Encoder

+) Bidirectional Encoder Representations from transformers
BERT
- Makes use of transfer learning / pretraining
- A multi layer bi-transformer
- Positional embeddings

• BERT - base

12 layers (12 transformer blocks)

12 attention heads

110 million params

masks 15% words

During pretraining, the model is trained on unlabeled data over different pre-training tasks

. Choose 15% of tokens at random

mask 80% of the time, replace with random

keep as is 10% , token 10%

. Multiple masked spans in a sentence

. Next sentence prediction

for finetuning, initialized Bert with pretrained params, using labeled data from the downstream tasks.

+) BERT objective

combine word embeddings, sentence embeddings, positional embeddings

Formatting the input

Input

+ Token Embeddings
+ Segment Embeddings
+ Position Embeddings

[CLS] ... [SEP] ... [SEP]

C $T_1 \dots T_n$ [sep] $T_1' \dots T_m'$

↓ NSP ↓ mask LM ↓ mask LM

- Objective 1:
  Multi mask LM
  Loss: Cross Entropy Loss

- Objective 2:
  Next Sentence Prediction
  Loss: Binary loss
  (Do they follow or Not)

+) Fine tuning BERT

Sentence A	Sentence B
text	ϕ
Question	Passage
Hypothesis	Premise
Sentence	Entities
Sentence	Paraphrase
Article	Summary

+) T5 : Transfer learning + Masked language modeling

< >

Base

Encoder · Decoder

↓ ↓

full bidirectional attention | causal attention

Language Model

causal masking

(concatenate inputs and target)

Prefix LM

full bidirectional attention on input + Mix Causal

+) Multi-task training strategy

· Predict entailment, contradiction, or neutral

· Winograd schema + +

Training data strategies:

Examples-proportional mixing, temperature-scaled mixing

Equal mixing

Fine-tuning method:

+ All params

Adapter layers

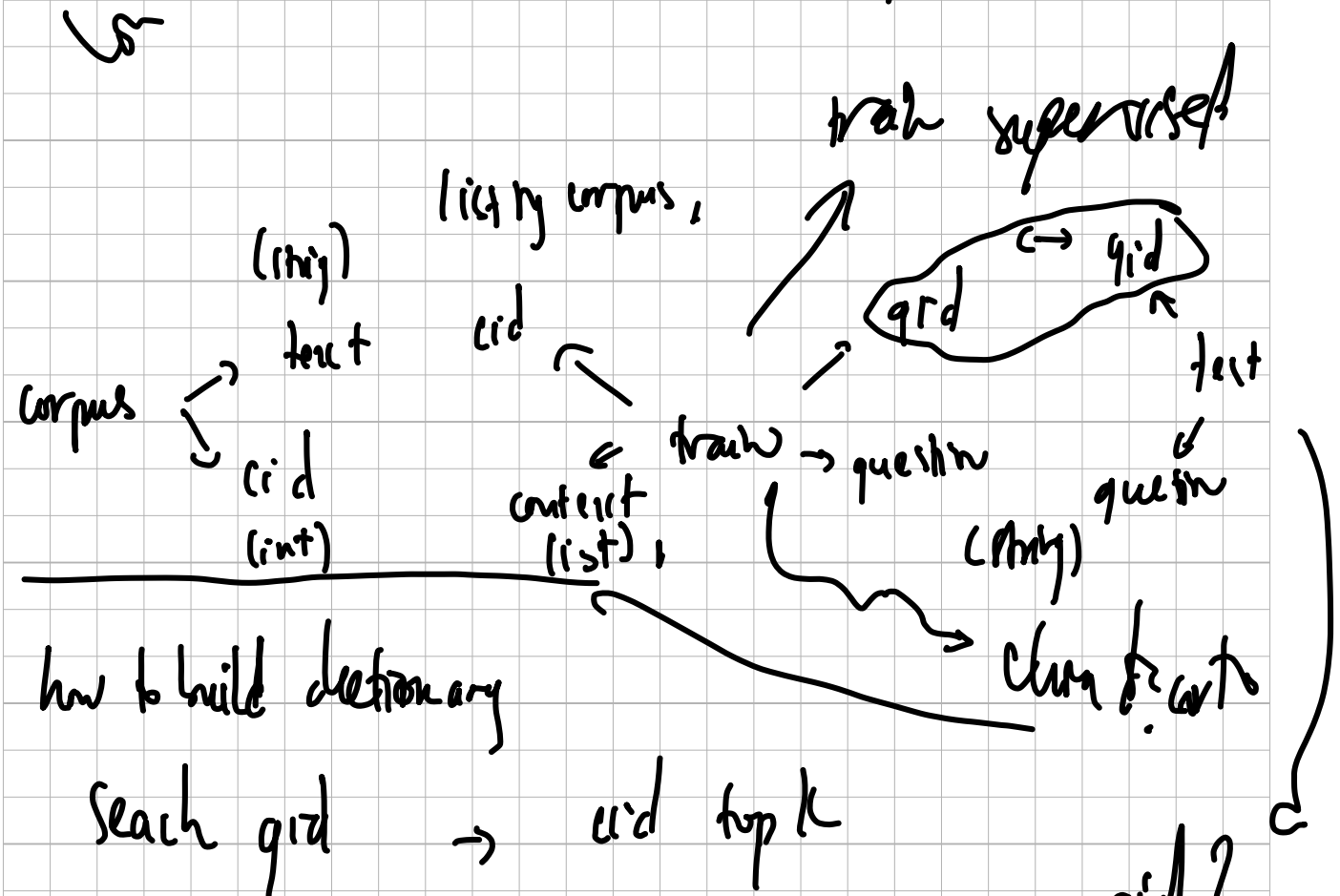
Gradual unfreezing

+) GLUE benchmark

+) Hugging Face : using transformers

Pipelines

1. Pre-processing your inputs
2. Running the model
3. Post-processing the outputs



phoBert pyvncorelp fair seg

sentence-bi-encoder