N-gram language modeling:

distributions are sentences

$P(\bar{W})$ ← sequence of words $= P(w_1, \cdots, w_m) = P(w_1) P(w_2 | w_1) P(w_3 | w_1, w_2) \cdots$

n-gram LM

$$P(\bar{W}) = \prod_{i=1}^{m} P(w_i | \underbrace{w_{i-n+1}, \cdots, w_{i-1}}_{\text{prev. } n-1 \text{ words}})$$

2-gram LM: $P(w_1 | <s>) P(w_2 | w_1) P(w_3 | w_2) \cdots$

n-gram LM <=> n-1 order Markov model

3-gram LM: $P(w_1 | <s><s>) P(w_2 | <s> w_1) P(w_3 | w_1 w_2) \cdots$

2-gram: multinomial distribution $|V| \times |V|$ params

↓ table of probabilities

$P(w | the) =$ 0.001 house (very flat list.)
0.005 dog
0.005 cat

Param estimation: MLE (max likelihood estimation) from a large corpus

$$P(dog | the) = \frac{count(the, dog)}{count(the)}$$

Why ① Generation: Machine translation
② Grammatical error correction
③ way to build "word2vec++"

Smoothing in n-gram LMs:

5-gram models work well

P(Austin | to) > 0 seen in data

P(Austin | want to go to) = 0 if corpus isn't huge

Smoothing fixes this → gives some probability to unseen instances

Absolute Discounting:

reserve mass from seen 5grams to allocate to unseen 5 grams

$$P(\text{Austin} \mid \text{want to go to}) = \frac{\text{count}(\text{wtgtA})}{\text{count}(\text{wtgt})} + \lambda P_{AD}(A \mid tgt)$$

λ is set to make this normalize

want to go to
- → Maui 2 → 1.8
- → Class 1 → 0.8
- → campus 1 → 0.8

count = 4

k = 0.2

$\lambda = \frac{0.6}{4}$ (# of unique words seen in this context) times k

$$P_{AD}(A \mid tgt) = \cdots + \lambda' P_{AD}(A \mid gt)$$ recursive

→ → P(A) > 0 — if Austin in our data

Kneser-Ney smoothing

LM Evaluation:

regular accuracy doesn't make sense as a metric in LMs

instead use Likelihood on held out data

$$\frac{1}{n}\sum_{i=1}^{n} \log P(w_i | w_{i,} \cdots, w_{i-1}) \quad \text{— log likelihood}$$

perplexity: exp(avg NLL) — lower = better

suppose probs: 1/4, 1/3, 1/4, 1/3

avg NLL (base e) = 1.242 Perplexity = 3.464 ← uses geometric mean of denominator

ranges from 10–200 standard n LMs but not really elsewhere

Neural Language Models:

$$P(\bar{w}) = P(w_1)\ P(w_2|w_1) \cdots P(w_n | w_1 \cdots w_{n-1})$$

$P(w_n | w_1 \cdots w_{n-1})$: model with NN

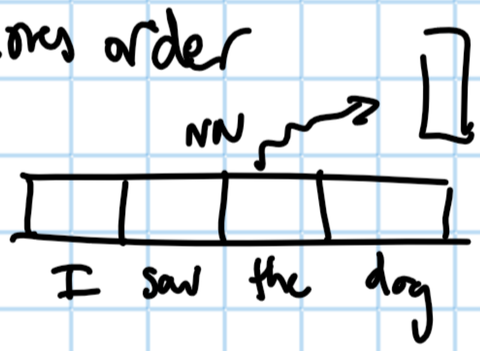
basic: $P(w_i | w_{i-1}) = \frac{e^{v_i \cdot c_{i-1}}}{\sum_{w' \in V} e^{v_w \cdot c_{i-1}}}$

more generally: $P(w_i | w_1 \cdots w_{i-1}) = \text{softmax}(U_{w_i} \cdot f(w_1 \cdots w_{i-1}))$

f: NN to embed context

f = DAN → ignores order

f = FFNN

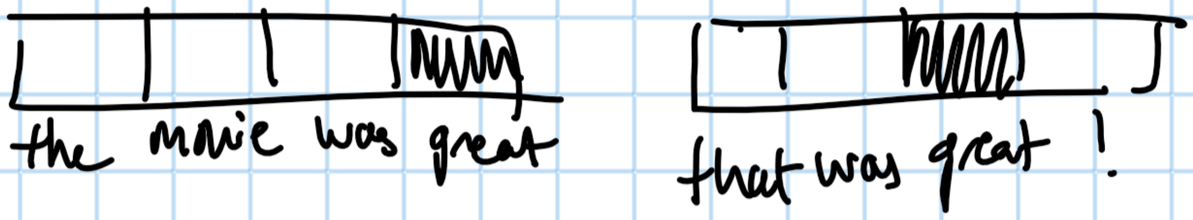


matmul + softmax → $P(w_i | w_{1,} \cdots, w_{i-1})$

Main + Hinton (2003) tried it out but this does it scale to long context (same as n-gram LM)

RNN's and their Shortcomings:

FFNN's can't handle variable length inputs: each position in the feature vector has fixed semantics



these are not related (great will be in 2 different orthogonal subspaces)

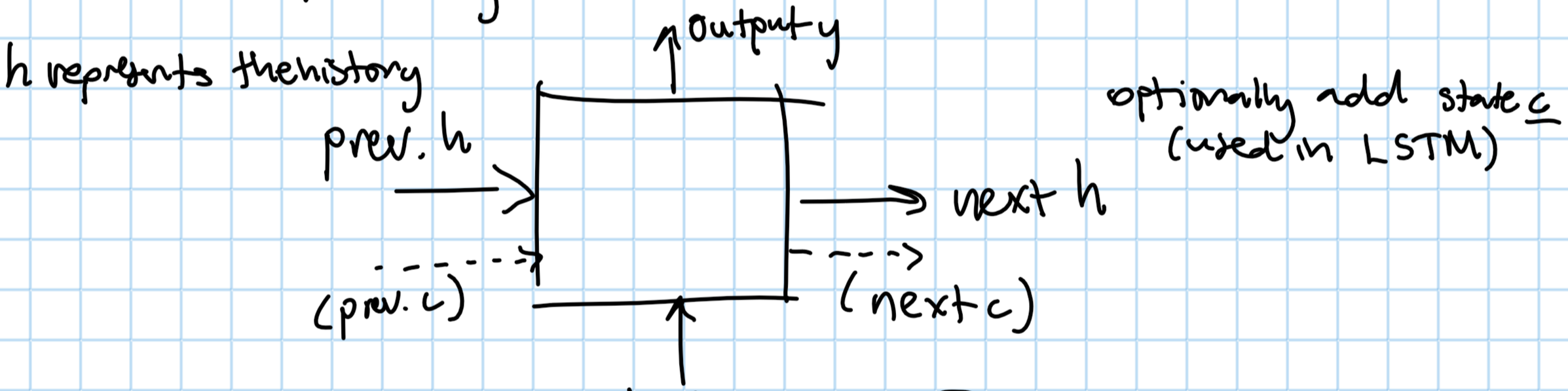
DAN has opposite problem, where all position is gone

instead:

① process each word uniformly
② still using the context that the token occurs in

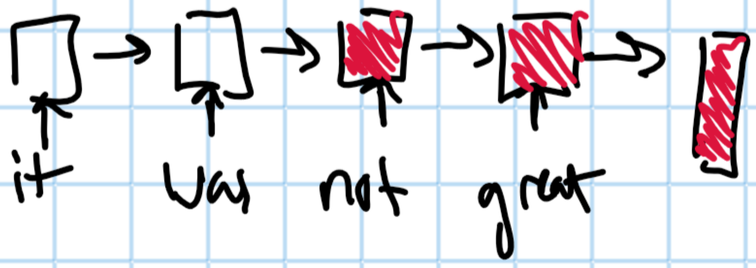
RNN:

cell that has input x, has hidden state h, and updates that hidden state and produces y (all vectors)

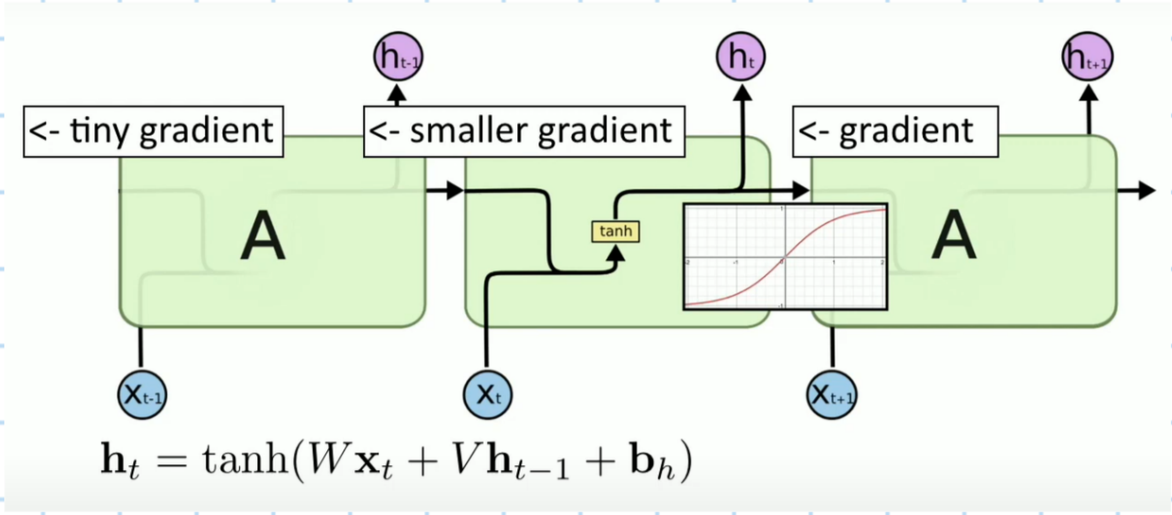


ex:

if blue means positive sentiment, the model recognizes great in any part of the sentence and flips the final output to positive



Vanishing Gradient:



if input is at ends of spectrum, you're going to lose gradient in backpropagation

if not in $[-2, 2]$ grad is almost 0

h repeated multip by V can cause it to blow up or shrink depending on

hard to get information from gradients in long scale networks

LSTM can help this, but not enough

RNNs are also slow. They don't parallelize (for GPU) and there are O(n) non-parallel operations for encoding n items

Solution: Transformers

Attention

ex: All As: last letter is A, any B = last letter is B

AAAAAAA
ABAAAAB
AAAAABB

Attention: method to go arbitrarily far back in context from this point

BAAA···AB ← As

→ RNNs struggle with this (remembering context for many positions is hard)

A A B A _ (B)

Keys: embeddings of seq.
Query: what we want to find

Assume: A=[1, 0] B=[0, 1]
call them e_i

Step 1: compute score for each key given query
score: $s_i = k_i^T q$

[1 0][0][0][1 0]
A A B A
0 0 1 0

Set $q = [0\ 1]$ find Bs

Step 2: Softmax

A A B A ⇒ softmax: (e=3)
0 0 1 0 → 1/6 1/6 1/2 1/6

Step 3: compute output as weighted sum of the input

result: $\sum_{i=1}^{n} \alpha_i e_i = \left(\frac{1}{6}[1\ 0] \cdot 3 + \frac{1}{2}[0\ 1]\right)$

$= [\frac{1}{2}, \frac{1}{2}]$

Compare to average: $[\frac{3}{4}\ \frac{1}{4}]$

We can amplify embeddings

$k_i = W^k e_i$ $W^k = \begin{matrix} 10 & 0 \\ 0 & 10 \end{matrix}$

[10 0][10 0][0 10][10 0]
0 0 10 0

softmax → $\left[\frac{1}{22026}, \frac{1}{22029}, \frac{22026}{22029}, \frac{1}{22029}\right]$

dot product attention: $s_i = k_i^T q$
scaled dpa: $s_i = k_i^T W_q$
equiv to 2 wt mat: $s_i = (W^k k_i)(W^q q)$

real attention uses 3 matrices (one for values)

Self Attention

builds an attention — every word in a sequence is both a key and a query simultaneously

Q: seq len x d mat (d = embedding dim = 2 in notes)
K: seq len x d mat

$$W^Q = \begin{matrix} 0 & 1 \\ 0 & 1 \end{matrix}$$ no matter value, look for Bs

$$W^k = \begin{matrix} 10 & 0 \\ 0 & 10 \end{matrix}$$ "booster"

multiple ways to set up weights that are equiv. but this has clear structure

$$E = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad Q = EW^Q = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \qquad K = EW^k = \begin{pmatrix} 10 & 0 \\ 10 & 0 \\ 0 & 10 \\ 10 & 0 \end{pmatrix}$$

scores: $S = QK^T$ $\quad S_{ij} = q_i \cdot k_j$

$$S = \begin{pmatrix} 0 & 0 & 10 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 10 & 0 \end{pmatrix}$$

rows represent attention scores

"I care about word 3"

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- n = sentence length, d = hidden dim, k = kernel size, r = restricted neighborhood size
- **Quadratic complexity**, but O(1) sequential operations (not linear like in RNNs) and O(1) "path" for words to inform each other

rowwise softmax => distribution per row

output = AE (actually $A(EW^v)$) ← value mat

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

S aka attention matrix

less peak

$V = EW^v$

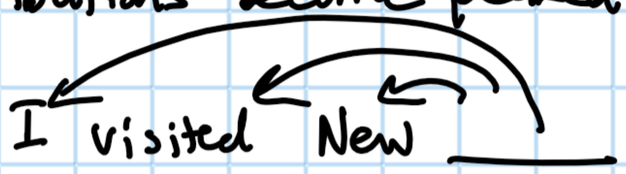
one head of self attention — multi heads via rand init params

produces
final mat * input = same dim as input
contextualized encoding for each word preserving length of sequence

input sequence of words → output vector same length of sequence that represents those words in context

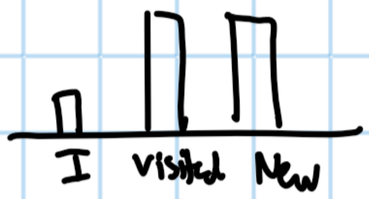
Multihead Self Attention

though attention can theoretically attend to multiple tokens, in practice softmax distributions become peaked

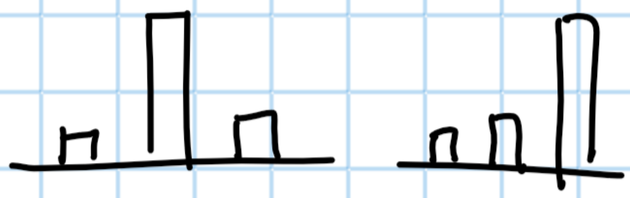


peaks on one or another

want:



get:



layers can help, but it would be great if we could do this in a single layer

Solution: multiple heads to do copies of attention

have 2 sets of random init matrices and compute twice

concatenate all resulting mats and multiply by a mat to bring it to output dim

Position Encodings

I visited New York and ate _____

right now, visited and ate could switch positions and we'd get the same results, unless we provide position information

encode each sequence position as an integer, add it to the word emb vector

affects input layer, so the model will learn how to attend to the 2 emb vectors

relative positional encoding: uses distance between 2 tokens

ALiBi: $\text{softmax}(q_i K^T + m[-(i-1), \ldots, -2, -1, 0])$

linear bias attend less for further tokens

each head will have different m

no position encodings also works with modified transformer

only looks at the past, so the model can learn position from new tokens