Language Models

He He

New York University

September 29, 2020

Table of Contents

Predict sequences

First part:

- Text representation ϕ: text $\rightarrow \mathbb{R}^d$
  - BoW representation
  - Distributed representation (word embeddings)
- Probabilistic models
  - Multinomial Naive Bayes
  - Logistic regression

Second part:

- Predict sequences
- Predict trees
- Inference algorithms

Language modeling

Motivation: pick the most probable sentence

- Speech recognition

  the *tail* of a dog
  the *tale* of a dog

  It's not easy to *wreck a nice beach*.
  It's not easy to *recognize speech*.
  It's not easy to *wreck an ice beach*.

- Machine translation

  He sat on the *table*.
  He sat on the *figure*.

  Such a Europe would *the rejection of any* ethnic nationalism.
  Such a Europe would *mark the refusal of all* ethnic nationalism.

Problem formulation

- **Vocabulary**: a *finite* set of symbols $\mathcal{V}$, e.g. $\{\text{fox}, \text{green}, \text{red}, \text{dreamed}, \text{jumped}, \text{a}, \text{the}\}$
- **Sentence**: a *finite* sequence over the vocabulary $x_1 x_2 \ldots x_n \in \mathcal{V}^n$ where $n \geq 0$ (empty sequence when $n = 0$)
- The set of all sentences: $\mathcal{V}^*$
- Goal: Assign a probability $p(x)$ to all sentences $x \in \mathcal{V}^*$.

Problem formulation

- **Vocabulary**: a *finite* set of symbols $\mathcal{V}$, e.g. $\{\text{fox}, \text{green}, \text{red}, \text{dreamed}, \text{jumped}, \text{a}, \text{the}\}$
- **Sentence**: a *finite* sequence over the vocabulary $x_1 x_2 \ldots x_n \in \mathcal{V}^n$ where $n \geq 0$ (empty sequence when $n = 0$)
- The set of all sentences: $\mathcal{V}^*$
- Goal: Assign a probability $p(x)$ to all sentences $x \in \mathcal{V}^*$.

Assign probabilities:

the fox jumped 0.01

the green fox dreamed 0.001

the green dreamed fox 0.0001

dreamed red fox the 0.0001

Table of Contents

Learning a LM

- Given a corpus consisting of a set of sentences: $D = \left\{x^{(i)}\right\}_{i=1}^{N}$
- Define

$$p_s(x) = \frac{\text{count(x)}}{N} .$$

(Check that $\sum_{x \in \mathcal{V}^*} p_s(x) = 1$.)

- Is p_s a good LM?

- estimation

- zero prob. on unseen data / generalization

Learning a LM

- Given a corpus consisting of a set of sentences: $D = \left\{x^{(i)}\right\}_{i=1}^{N}$
- Define
$$p_s(x) = \frac{\text{count(x)}}{N} .$$
(Check that $\sum_{x \in \mathcal{V}^*} p_s(x) = 1$.)
- Is p_s a good LM?

Need to reduce the number of model parameters.

Simplification 1: sentences to symbols

Decompose the joint probability using **chain rule**:

$$
\begin{aligned}
p(x) &= p(x_1, \ldots, x_n) \\
&= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \ldots p(x_n \mid x_1, \ldots, x_{n-1}) \\
&= p(x_n)p(x_{n-1} \mid x_n) \ldots p(x_1 \mid x_2 \ldots x_{n-1})
\end{aligned}
$$

left to right

Reduced number of outcomes: $\mathcal{V}^*$ (sentences) to $\mathcal{V}$ (symbols)

But there is still a large number of contexts!

Simplification 2: limited context

Reduce dependence on context by the **Markov assumption**:

- First-order Markov model

$$
\begin{aligned}
p(x_i \mid x_1, \ldots, x_{i-1}) &= p(x_i \mid x_{i-1}) \\
p(x) &= \prod_{i=1}^{n} p(x_i \mid x_{i-1})
\end{aligned}
$$

- Number of contexts: $|\mathcal{V}|$
- Number of parameters: $|\mathcal{V}|^2$

Beginning of a sequence:

$$
p(x_1 \mid x_{1-1}) = ?
$$

Assume sequence starts with a special start symbol: $x_0 = *$.

Model sequences of variable lengths

It
*

Sample a sequence from the first-order Markov model $p(x_i \mid x_{i-1})$:

* It is ...

When to stop?

Model sequences of variable lengths

Sample a sequence from the first-order Markov model $p(x_i \mid x_{i-1})$:

When to stop?

Assume that all sequences end with a stop symbol $\texttt{STOP}$, e.g.

$$
\begin{aligned}
& p(\text{the}, \text{fox}, \text{jumped}, \texttt{STOP}) \\
= & \, p(\text{the} \mid *) p(\text{fox} \mid \text{the}) p(\text{jumped} \mid \text{fox}) p(\texttt{STOP} \mid \text{jumped})
\end{aligned}
$$

LM with the $\texttt{STOP}$ symbol:

- Vocabulary: $\texttt{STOP} \in \mathcal{V}$
- Sentence: $x_1 x_2 \ldots x_n \in \mathcal{V}^n$ for $n \geq 1$ and $x_n = \texttt{STOP}$.

$x_1 = \text{STOP}$: empty sequence

N-gram LM

- Unigram language model:

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i) \; .$$

- Bigram language model ($x_0 = *$):

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_{i-1}) \; .$$

- Trigram language model ($x_{-1} = *, x_0 = *$):

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_{i-2}, x_{i-1}) \; .$$

- n-gram language model:

$$p(x_1, \ldots, x_m) = \prod_{i=1}^{m} p(x_i \mid \underbrace{x_{i-n+1}, \ldots, x_{i-1}}_{\text{previous } n-1 \text{ words}}) \; .$$

Parameter estimation

- Data: a corpus $\{x^{(i)}\}_{i=1}^{N}$ where $x \in \mathcal{V}^{n}$.
- Model: bigram LM $p(w \mid w')$ for $w, w' \in \mathcal{V}$.

$$p(w \mid w') = \theta_{w|w'}$$

where $\sum_{w\in\mathcal{V}} p(w \mid w') = 1 \quad \forall w' \in \mathcal{V}$.

MLE:

$$\max_{\theta \in \mathbb{R}^{|V|\times|V|}} \ell(\theta) = \sum_{i=1}^{N} \sum_{j=1}^{n} \log \underbrace{p(x_j \mid x_{j-1})}_{\theta_{x_j|x_{j-1}}}$$

$$\text{s.t. } \sum_{w\in V} \theta_{w|w'} = 1 \quad \forall w' \in V$$

MLE solution

- Unigram LM

$$\hat{p}(x) = \frac{\text{count}(w)}{\sum_{w \in \mathcal{V}} \text{count}(w)}$$

- Bigram LM

$$\hat{p}(w \mid w') = \frac{\text{count}(w, w')}{\sum_{w \in \mathcal{V}} \text{count}(w, w')} = \text{count}(w')$$

- In general, for n-gram LM,

$$\hat{p}(w \mid c) = \frac{\text{count}(w, c)}{\sum_{w \in \mathcal{V}} \text{count}(w, c)}$$

where $c \in \mathcal{V}^{n-1}$.

Example

- Training corpus

$$\{\text{The fox is red}, \text{The red fox jumped}, \text{I saw a red fox}\}$$

- Collect counts

$$\text{count}(\text{fox}) = 3$$
$$\text{count}(\text{red}) = 3$$
$$\text{count}(\text{red}, \text{fox}) = 2$$
$$\ldots$$

- Parameter estimates

$$\hat{p}(\text{red} \mid \text{fox}) = \frac{\text{count}(\text{red}, \text{fox})}{\text{count}(\text{fox})} = \frac{2}{3}$$
$$\hat{p}(\text{saw} \mid \text{i}) = \frac{1}{1} = 1$$

Example

- Training corpus

  $$\{\text{The fox is red}, \text{The red fox jumped}, \text{I saw a red fox}\}$$

- Collect counts

  $$\text{count}(\text{fox}) = 3$$
  $$\text{count}(\text{red}) = 3$$
  $$\text{count}(\text{red}, \text{fox}) = 2$$
  $$\ldots$$

- Parameter estimates

  $$\hat{p}(\text{red} \mid \text{fox}) =$$
  $$\hat{p}(\text{saw} \mid \text{i}) =$$

- What is the probability of "I saw a brown fox jumped"? $= 0$

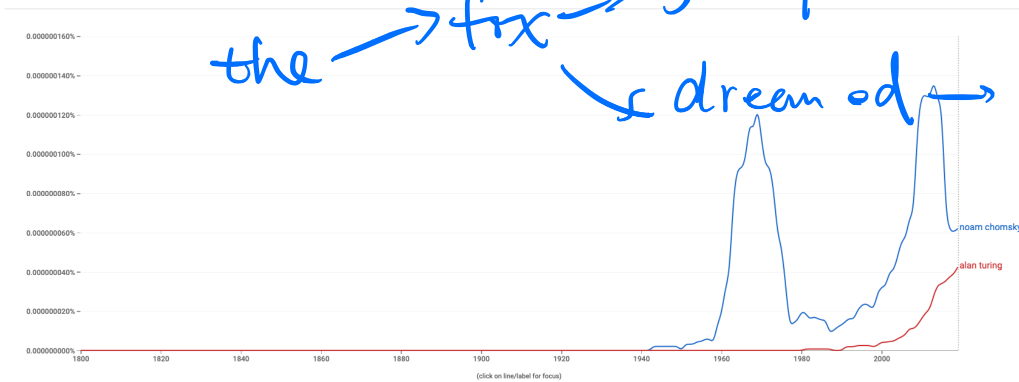
Real n-gram counts

Google Books n-gram counts

Trie

the → fox → jumped → val

↘ dreamed → val



context encoding

word	context	value

64 bit

Efficient implementation

- Memory, inference speed
- Context encodings, tries, caching, ...
- kenlm (`https://github.com/kpu/kenlm`)

Summary

Language models: assign probabilities to sentences

N-gram language models:

- Assume each word only conditions on the previous $n - 1$ words
- MLE estimate: counting n-grams in the training corpus

Problems with vanilla n-gram models:

- Estimate of probabilities involving rare n-grams is inaccurate
- Sentences containing unseen n-grams have zero probability

Backoff and interpolation

What context size should we use?

Backoff: Use higher-order models when we have enough evidence.

- Stupid backoff:

$$
S\cancel{\hat{p}}(x_i \mid x_{i-n+1:i-1}) = \begin{cases} \frac{\text{count}(x_{i-n+1:i})}{\text{count}(x_{i-n+1:i-1})} & \text{if count}(x_{i-n+1:i}) > 0 \\ \lambda \underset{S}{\cancel{\hat{p}}}(x_i \mid x_{i-n+2:i-1}) & \text{otherwise} \end{cases}
$$

Interpolation: mixture of n-gram models

$$
p(x_i \mid x_{i-2}, x_{i-1}) = \lambda_1 p(x_i \mid x_{i-2}, x_{i-1}) + \lambda_2 p(x_i \mid x_{i-1}) + \lambda_3 p(x_i)
$$

where $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

- λ can depend on context. $\lambda_1(x_{i-2}, x_{i-1})$, $\lambda_2(x_{i-1})$
- Tune λ's on the validation set.

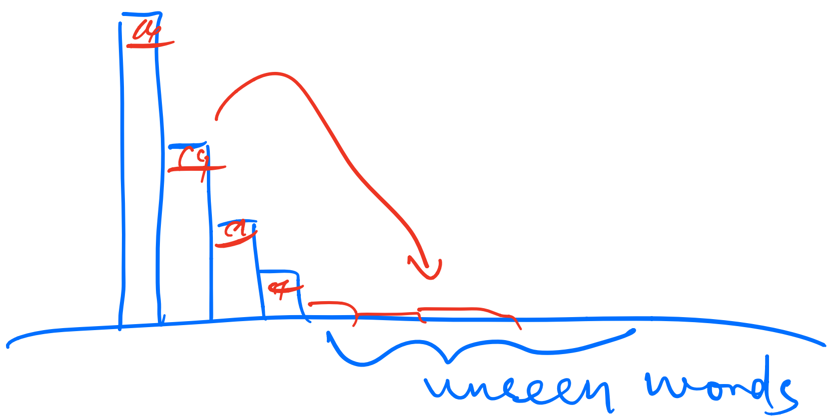
Smoothing

How to estimate frequencies of unseen words?

More generally, estimate unseen elements in the support of a distribution.

- Given frequencies of observed species, what's the probability of encountering a new species?
- Given observed genetic variations from a certain population, what's the probability of observing new mutations?

Key idea: reserve some probability mass for unseen words



Add-α smoothing

Original estimate: $\dfrac{\text{count}(x)}{N}$

Smoothed estiamte: $\dfrac{\text{count}(x)+\alpha}{N+\alpha|V|}$ → pseudo count

Discounted counts:

$$\frac{\text{count}^*(x)}{N} = \frac{\text{count}(x)+\alpha}{N+\alpha|V|}$$

$$\text{count}^*(x) = (\text{count}(x)+\alpha)\frac{N}{N+\alpha|V|}$$

Add-one smoothing

How does smoothing change the estimate?

Example:

$\text{count}(x) = 10, N = 100, |\mathcal{V}| = 1000$

Original: $10/100 = 0.1$

$1 \times |V|$

Smoothed: $(10 + 1)/(100 + 1000) \approx 0.01$

Assigns too much probability mass to unseen words!

Tuning α on validation set helps but still not good enough for LM.

Good-Turing smoothing

Key idea: use the validation set for estimation

train val

count(x) $\longrightarrow$ count*(x)

Leave-one-out cross validation

1 2 ... k (k-folds)

train val

$\underbrace{\qquad}_{N-1 \text{ examples}}$ $\underbrace{\qquad}_{1 \text{ examples}}$

1
2
⋮
k

count$_i$(x) count$^*_i$(x)

Good-Turing smoothing

- Let N_r be the number of tokens that occur r times in the corpus
- How many held-out tokens are unseen during training? N_1

train val

M $c(x)=0$ $c(x)=1$

- How many held-out tokens are seen k times during training?

$(k+1)N_{k+1}$

- What's the "correct" count of a word that occur k times in the corpus?

$$\text{count}^*(x) = \frac{(k+1)N_{k+1}}{N_k}$$

counts in held-out

words occur k time

- What's the probability of a word that occur k times in training?

$$\hat{p}_k = \frac{\text{count}^*(x)}{M} = \frac{(k+1)N_{k+1}}{MN_k}$$

$$\hat{p}_0 = \frac{N_1}{M}$$

Kneser-Ney smoothing

Widely used for n-gram LMs.

Idea 1: absolute discounting.

held-out

Count in 22M Words	Avg in Next 22M	Good-Turing c*
1	0.448	0.446
2	1.25	1.26
3	2.24	2.24
4	3.23	3.24

Figure: Good-Turing counts from Dan Klein's slides

Just subtract 0.75 or some constant.

Kneser-Ney smoothing

Idea 2: consider word versatility rather than word counts.

Motivation:

count(San Francisco) $= 100$, count(Minneapolis) $= 10$

I recently visited ______.

Kneser-Ney smoothing

Idea 2: consider word versatility rather than word counts.

Motivation:

count(San Francisco) = 100, count(Minneapolis) = 10

I recently visited ______.

Some words can only follow specific contexts, i.e. less versatile.

Continuation probability: how likely is w allowed in a context

$p_{\text{unigram}}(w) \propto \sum_{w' \in \mathcal{V}} \text{count}(w, w')$

$p_{\text{continuation}}(w) \propto |\{w' \colon \text{count}(w, w') > 0\}|$

$$\beta(w) = \frac{\#\text{ bigram types ends with } w}{\#\text{ bigram types}}$$

Kneser-Ney smoothing

Combine the two ideas:

0.75

versatility

$$\hat{p}(w \mid w') = \frac{\max(\text{count}(w, w') - d, 0)}{\text{count(w')}} + \lambda(w')p_{\text{continuation}}(w)$$

abs. discount

interpolation

- Works well for ASR and MT.
- Dominating n-gram model before neural LMs.

Summary

Key ideas in n-gram language models:

Markov assumption:

- Trigram models are reasonable.
- ASR, MT often use 4- or 5-gram models.

Discounting / Smoothing:

- "Borrow" probability mass for unseen words
- Good-Turing smoothing, absolute discount

Dynamic context:

- Use more context if there is evidence
- Katz backoff, Kneser-Ney

See Chen and Goodman (1999) for more results.

Table of Contents

N-gram models by classification

Log-linear language model: $\theta_w \cdot \phi(c)$

output

context

$$p(w \mid c) = \frac{\exp\left[\theta \cdot \phi(w, c)\right]}{\sum_{w' \in \mathcal{V}} \exp\left[\theta \cdot \phi(w', c)\right]}$$

Feature templates:

$T_1(w, c) = w, c[-1]$

$T_2(w, c) = w, \text{POS}(c[-1])$

$T_3(w, c) = w, c[-1], c[-2]$

$\vdots$

Features:

- Data:
  The brown fox jumped
- $\phi_1(w, c)$
  $= \mathbb{1}(w = \text{the}, c[-1] = *)$
  $\phi_2(w, c)$
  $= \mathbb{1}(w = \text{brown}, c[-1] = \text{the})$
  $\vdots$

Learn by MLE and SGD.

Feed-forward neural networks

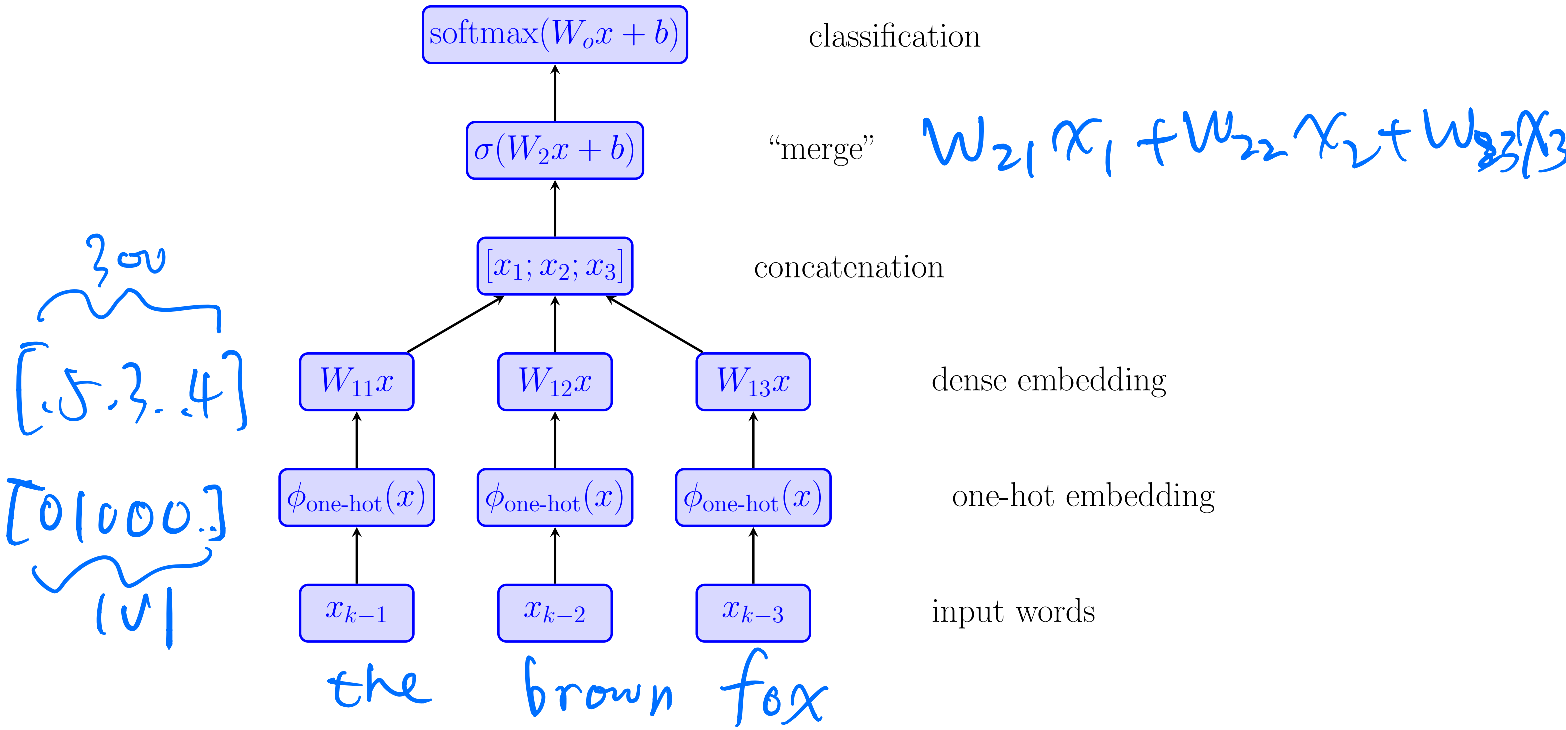
Key idea in neural nets: feature/representation learning

Building blocks:

- Input layer: raw features (no learnable parameters)
- Hidden layer: perceptron + nonlinear activation function
- Output layer: linear (+ transformation, e.g. softmax)

Feed-forward neural language models

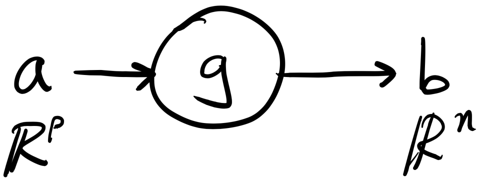
Encode the (fixed-length) context using feed-forward NN:



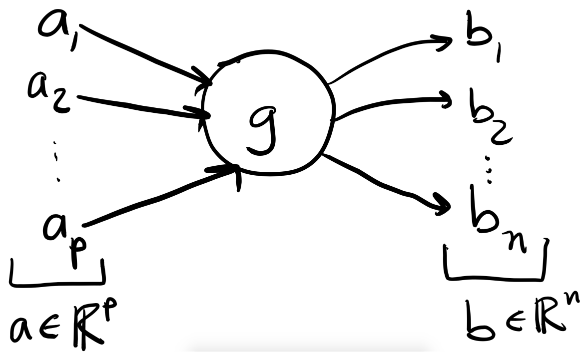
Computation graphs

Function as a node that takes in inputs and produces outputs.

- Typical computation graph:



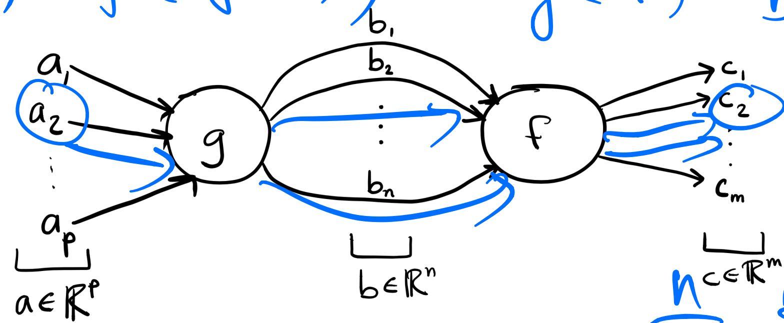
- Broken out into components:



Compose multiple functions

Compose two functions $g : \mathbb{R}^p \rightarrow \mathbb{R}^n$ and $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$.

$$\vec{c} = f \circ g(\vec{a}) = f(g(\vec{a})) \qquad g(\vec{a}) = \vec{b}$$



$$= \sum_{k=1}^{n} \frac{\partial c_i}{\partial b_k} \frac{\partial b_k}{\partial a_j}$$

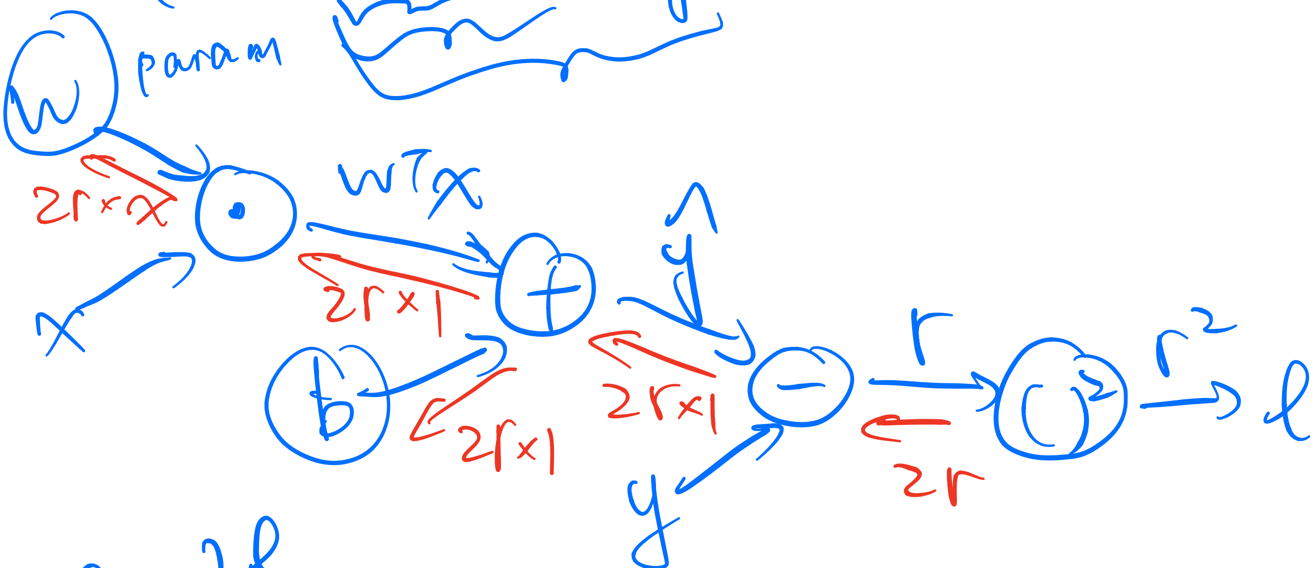
- How does change in a_j affect c_i?
- Visualize **chain rule**:
  - Sum changes induced on all paths from a_j to c_i.
  - Changes on one path is the product of changes on each edge.

$$\frac{\partial c_i}{\partial a_j} = \frac{\partial c_i}{\partial \vec{b}} \cdot \frac{\partial \vec{b}}{\partial a_j}$$

$$\frac{\partial c_i}{\partial a_j} = \sum_{k=1}^{n} \frac{\partial c_i}{\partial b_k} \frac{\partial b_k}{\partial a_j}.$$

Computation graph example

$$\ell = (w^T x + b - y)^2 \qquad \ell = r^2$$



① $\dfrac{\partial \ell}{\partial r} = 2r$

② $\dfrac{\partial \ell}{\partial \hat{y}} = \dfrac{\partial \ell}{\partial r} \cdot \dfrac{\partial r}{\partial \hat{y}} = 2r \times 1 = 2r$

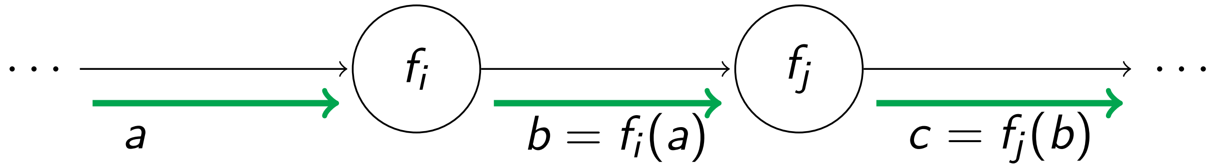
③ $\dfrac{\partial \ell}{\partial w} = \dfrac{\partial \ell}{\partial r} \cdot \dfrac{\partial r}{\partial \hat{y}} \cdot \dfrac{\partial \hat{y}}{\partial w} = 2rx$

Backpropogation

Backpropogation = chain rule + dynamic programming on a computation graph

Forward pass

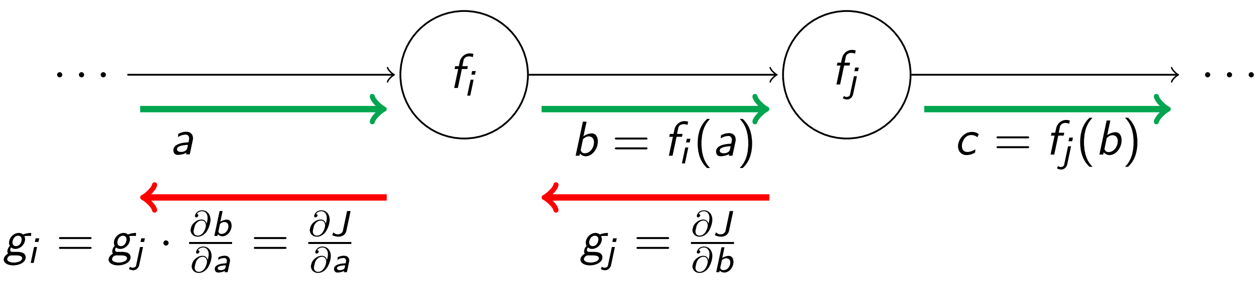
- **Topological order**: every node appears before its children
- For each node, compute the output given the input (from its parents).



Backpropogation

Backward pass

- **Reverse topological order**: every node appear after its children
- For each node, compute the partial derivative of its output w.r.t. its input, multiplied by the partial derivative from its children (chain rule).



Summary

Neural networks

- Automatically learn the features
- Optimize by SGD (implemented by back-propogation)
- Non-convex, may not reach a global minimum

Feed-forward neural language models

- Use fixed-size context (similar to n-gram models)
- Represent context by feed-forward neural networks

Table of Contents

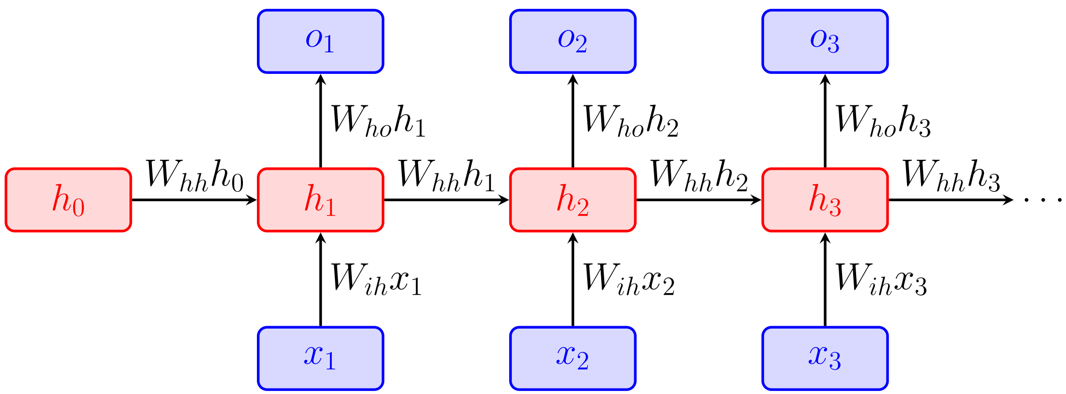
Recurrent neural networks

How much context is needed?

... I went to ______ ______

Idea: compute context representation recurrently

$$h_t = \sigma(\underbrace{W_{hh}h_{t-1}}_{\text{previous state}} + \underbrace{W_{ih}x_t}_{\text{new input}} + b_h) \; .$$



Backpropogation through time

$$\frac{\partial \ell}{\partial W} = \frac{\partial \ell}{\partial y_t} \frac{\partial y_t}{\partial f} \frac{\partial f}{\partial h_t} \frac{\partial h_t}{\partial W}$$

Exercise: compute $\frac{\partial h_t}{\partial h_i}$

$$h_t = \sigma(\underbrace{W_{hh}h_{t-1}}_{\text{previous state}} + \underbrace{W_{ih}x_t}_{\text{new input}} + b_h) \cdot \sum_{i=1}^{t} \frac{\partial h_t}{\partial h_i}$$

$$y_t = f(h_t)$$

Problem:

- Gradient involves repeated multiplication of $W_{hh} = Q \Lambda^k Q^T$
- Gradient will vanish / explode

Quick fixes:

- Truncate after k steps (i.e. `detach` in the backward pass)
- Gradient clipping

Gated recurrent neural networks

Long-short term memory (LSTM)

- Memory cell: decide when to "memorize" or "forget" a state

$\frac{\partial c_t}{\partial c_{t-1}}$

$$c_t = \underbrace{\overset{[0,1]}{i_t} \odot \tilde{c}_t}_{\text{update with new memory}} + \underbrace{\overset{[0,1]}{f_t} \odot c_{t-1}}_{\text{reset old memory}}$$

$$\tilde{c}_t = \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \ .$$

- Input gate and forget gate

$$i_t = \text{sigmoid}(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \ ,$$
$$f_t = \text{sigmoid}(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \ .$$

- Hidden state

$$h_t = o_t \odot c_t \ , \text{ where}$$
$$o_t = \text{sigmoid}(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \ .$$

Table of Contents

Perplexity

What is the loss function for learning language models?

Held-out likelihood on test data D:

$$\ell(D) = \sum_{i=1}^{|D|} \log p_\theta(x_i \mid x_{1:i-1}) \; ,$$

Perplexity:

$$\mathrm{PPL}(D) = 2^{-\frac{\ell(D)}{|D|}} \; .$$

avg NLL loss on test data

true dist

model

$2^{H(p)}$

- Cross entropy: $H(p, p_\theta) = -\mathbb{E}_{x \sim p} \log p_\theta(x)$.
- Interpretation: a model of perplexity k predicts the next word by throwing a fair k-sided die.