Language Models

He He

New York University

September 29, 2022

Table of Contents

Logistics

- HW1 due tonight 11:55pm
- HW2 released today

Last week

Goal: Learning useful representions of words

Distributional hypothesis: Words that occur in similar contexts tend to have similar meanings.

Methods

- Vector space models: infer clusters from co-occurrence statistics
- Self-supervised learning: predict parts of the text (e.g., words) from its context (e.g., neighbors)
- Brown clustering: find word classes in a hierarchical way

Basics of neural networks

- Learning intermediate subproblems and representations
- Activation functions allow for nonlinearity
- Optimize by backpropogation (today)

Predict sequences

First part:

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

Second part:

- Predict sequences
- Predict trees

Today: probabilistic modeling of sequences

Language modeling

Motivation: pick the most probable sentence from multiple hypothesis

- 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*.

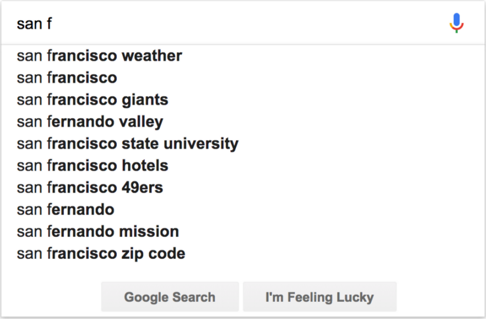
- Machine translation

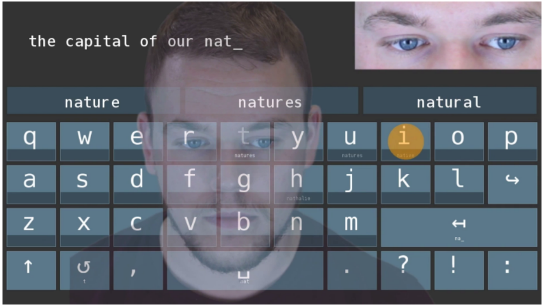
  He sat on the *table*.
  He sat on the *figure*.

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

Language modeling

Application: predict/suggest the next word





Problem formulation

Assign probabilities to a sequence of tokens:

$p(\text{the red fox jumped}) \gg p(\text{the green fox jumped})$

$p(\text{colorless green ideas sleep furiously}) \gg p(\text{furiously sleep ideas green colorless})$

- **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 (of different lengths): $\mathcal{V}^*$
- Goal: Assign a probability $p(x)$ to all sentences $x \in \mathcal{V}^*$.

Table of Contents

Learning a LM

- Given a corpus consisting of a set of sentences: $D = \left\{x^{(i)}\right\}_{i=1}^{N}$
  - Notation: $x_{\text{token id}}^{\text{(instance id)}}$
- Consider a multinomial distribution of sentences

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

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

- Is p_s a good LM?
  - Does not generalize to unseen data.
  - Need to restrict the model.

Simplification 1: sentence to tokens

Solve a smaller problem: model probability of each token

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

$$
\begin{aligned}
p(x) &= p(x_1, \ldots, x_n) \\
&= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \ldots p(x_n \mid x_1, \ldots, x_{n-1}) \\
&\text{(Doesn't have to go from left to right)} \\
&= p(x_n)p(x_{n-1} \mid x_n) \ldots p(x_1 \mid x_2, \ldots, x_n)
\end{aligned}
$$

- Problem reduced to modeling conditional token probabilities
- This is a classification problem we have seen
- 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

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

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

Beginning of a sequence:

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

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

Model sequences of variable lengths

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

1. Initial condition: $\text{prev} = *$
2. Iterate:
   2.1 $\text{curr} \sim p(\text{curr} \mid \text{prev})$
   2.2 $\text{prev} \leftarrow \text{curr}$

When to stop?

Assume that all sequences end with a **stop symbol** `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 `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 $x_n = \texttt{STOP}$.

N-gram LM

- Unigram language model (no context):

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

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

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

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

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

- n-gram language model:

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

Practical issues

- Use $n-1$ start symbols for n-gram models
- Trigram features are often used in classification
- Higher-order models (4/5-grams) are used for MT when large corpus is available
- All computation is done in the log space to avoid underflow

Parameter estimation

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

$$p(w \mid w') = \theta_{w|w'} \quad \text{multinomial distribution}$$

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

MLE: (HW2 P1)
[board]

MLE solution

- Unigram LM

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

- Bigram LM

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

- In general, for n-gram LM,

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

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

Example

- Training corpus (after tokenization)

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

- Collect counts

  $$\begin{aligned}&\text{count}(\text{fox}) = 3\\&\text{count}(\text{red}) = 3\\&\text{count}(\text{red}, \text{fox}) = 2\\&\ldots\end{aligned}$$

- Parameter estimates

  $$\begin{aligned}&\hat{p}(\text{red} \mid \text{fox}) = 2/3\\&\hat{p}(\text{saw} \mid \text{i}) = 1/1\end{aligned}$$

- What is the probability of "I saw a brown fox jumped"? Zero!
- What is the probability of "The fox saw I jumped"? Zero!

Summary so far

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

Out-of-vocabulary (OOV) words

Dealing with OOV words:

1. Choose a fixed vocabulary (e.g., all words in the training corpus that occur for more than 5 times)
2. Replace all OOV words (during training and test) by `<UNK>`
3. Treat `<UNK>` as a normal word

Smoothing

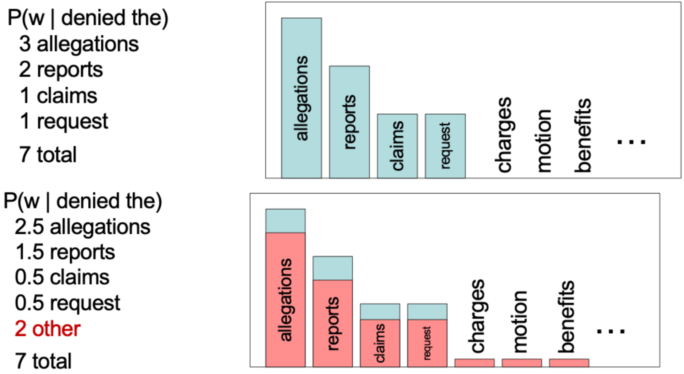
How to estimate frequencies of unseen words/n-grams?

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?

Smoothing

Key idea: reserve some probability mass for unseen words (discounting!)



(Figures from Dan Klein and John DeNero)

Add-α smoothing

Original estimate:

$$\frac{\text{count}(x)}{N}$$

Smoothed estiamte (add pseudo count to each word):

$$\frac{\text{count}(x) + \alpha}{N + \alpha|\mathcal{V}|}$$

Discounted counts:

$$\frac{\text{count}^*(x)}{N} = \frac{\text{count}(x) + \alpha}{N + \alpha|\mathcal{V}|} \tag{1}$$

$$\text{count}^*(x) = \frac{N}{N + \alpha|\mathcal{V}|}(\text{count}(x) + \alpha) \tag{2}$$

Add-one smoothing

How does smoothing change the estimate?

Example:

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

Original: $10/100 = 0.1$

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 in practice.

Good-Turing smoothing

Key idea: use a held-out (validation) set to estimate the "correct" counts and adjust the raw count accordingly

Leave-one-out cross validation

[board]

Good-Turing smoothing

- Let M be the total number of tokens
- Let N_r be the number of word types that occur r times in the corpus
- How many held-out tokens are unseen during training? N_1
- How many held-out tokens are seen k times during training? $N_{k+1}(k+1)$
- What's the "correct" count of a word that occur k times in the corpus?

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

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

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

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

Backoff

Problem: Cannot estiamte probability of rare n-grams accurately

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

First try:

$$p_{\text{backoff}}(x_i \mid x_{i-n+1:i-1}) = \begin{cases} p_{\text{MLE}}(x_i \mid x_{i-n+1:i-1}) & \text{if count}(x_{i-n+1:i}) > 0 \\ \alpha p_{\text{backoff}}(x_i \mid x_{i-n+2:i-1}) & \text{otherwise} \end{cases}$$

- If the n-gram has occured in the corpus, use the MLE estimate
- Otherwise, backoff to the $n-1$ gram estimate recursively with a constant backoff factor
- Not a proper probability distribution because of additional probability mass on unseen n-grams
- But works well in practice (Stupid Backoff [Brants et al., 2007])

Backoff

Problem: Cannot estiamte probability of rare n-grams accurately

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

Second try (Katz Backoff):

$$p_{\text{backoff}}(x_i \mid x_{i-n+1:i-1}) = \begin{cases} p_{\text{discount}}(x_i \mid x_{i-n+1:i-1}) & \text{if } \operatorname{count}(x_{i-n+1:i}) > 0 \\ \alpha(x_{i-n+1:i-1})\, p_{\text{backoff}}(x_i \mid x_{i-n+2:i-1}) & \text{otherwise} \end{cases}$$

- Discounted probability: reserve some probability mass for unseen events
- Backoff factor: probability mass distributed to unseen events given a specific context

Interpolation

Instead of backing off to lower-order models, we can use a 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 $\lambda_1 + \lambda_2 + \lambda_3 = 1$ (why?).

- λ can depend on context: $\lambda(x_{i-2}, x_{i-1})$.
- Tune λ's on the validation set.
- Model λ as a latent variable and solve by EM algorithm (later)

Kneser-Ney smoothing

Widely used for n-gram LMs.

Idea 1: absolute discounting.

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(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 c

$$
\begin{aligned}
p_{\text{continuation}}(w) &\propto |\{w' : \text{count}(w', w) > 0\}| \quad \# \text{ of context } w \text{ can follow} \\
&= \frac{|\{w' : \text{count}(w', w) > 0\}|}{\sum_w |\{w' : \text{count}(w', w) > 0\}|} \\
&= \frac{\# \text{ bigram types ends with } w}{\# \text{ bigram types}}
\end{aligned}
$$

Kneser-Ney smoothing

Combine the two ideas: absolute discount and continuation probability

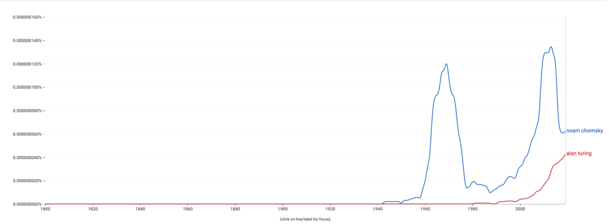
For bigrams:

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

- λ: discounted probability mass
- Works well for ASR and MT.
- Dominating n-gram model before neural LMs.

Real n-gram counts

Google Books n-gram counts



Efficient implementation

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

Summary

Key ideas in n-gram language models to handle sparsity:

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:

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

- Predict the output word given the context, e.g., $c =$ the brown fox and $w =$ jumped
- Use compatibility scores: $\theta_w \cdot \phi(c) \to \theta \cdot \phi(w, c)$

N-gram models by classification

How to design the feature map $\phi(w, c)$?

Corpus: "the brown fox jumped"

1. Define feature templates:
$$T_1(w, c) = (w, c[-1]) \text{ (bigram feature)}$$
$$T_2(w, c) = (w, \text{POS}(c[-1]))$$
$$T_3(w, c) = (w, \text{suffix}(c[-1]))$$
2. Read off features from the data
$$\phi_1(w, c) = \mathbb{I}(w = \text{the}, c[-1] = *)$$
$$\phi_2(w, c) = \mathbb{I}(w = \text{brown}, c[-1] = \text{the})$$

- Each template can produce many features
- Each class (word) has different features

Feed-forward neural networks

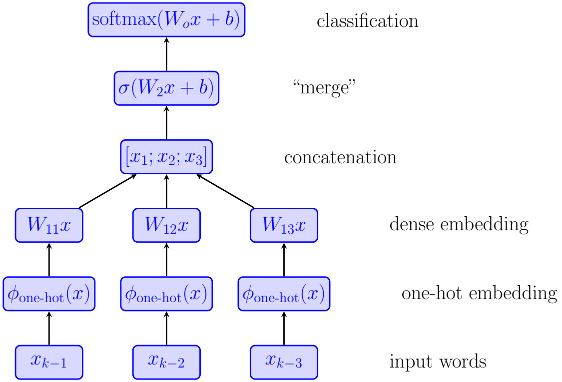
Key idea in neural nets: feature/representation learning

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:

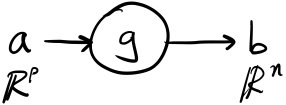


What kind of features may be learned?

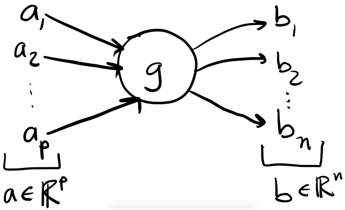
Computation graphs

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

- Typical computation graph:

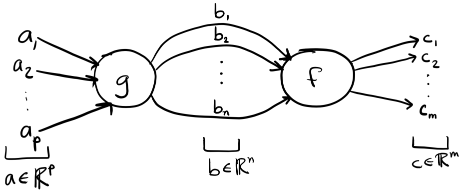


- Broken out into components:



Compose multiple functions

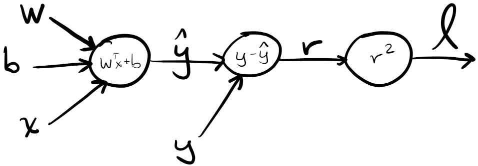
Compose two functions $g : \mathbb{R}^p \to \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}^m$.



- $c = f(g(a))$
- Derivative: How does change in a_j affect c_i?
- Visualize the **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.

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

Computation graph example



$$
\begin{aligned}
\frac{\partial \ell}{\partial r} &= 2r \\
\frac{\partial \ell}{\partial \hat{y}} &= \frac{\partial \ell}{\partial r}\frac{\partial r}{\partial \hat{y}} = (2r)(-1) = -2r \\
\frac{\partial \ell}{\partial b} &= \frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial b} = (-2r)(1) = -2r \\
\frac{\partial \ell}{\partial w_j} &= \frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_j} = (-2r)x_j = -2rx_j
\end{aligned}
$$

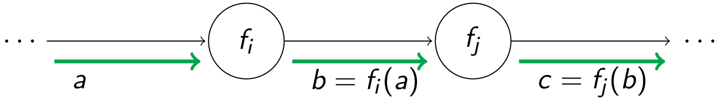
Example from David Rosenberg.

Backpropogation

Backpropogation = chain rule + dynamic programming on a computation graph

Forward pass

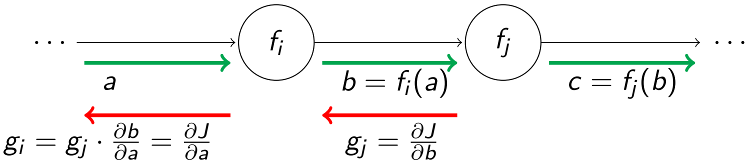
- **Topological order**: every node appears before its children
- For each node, compute the output given the input (from its parents).



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

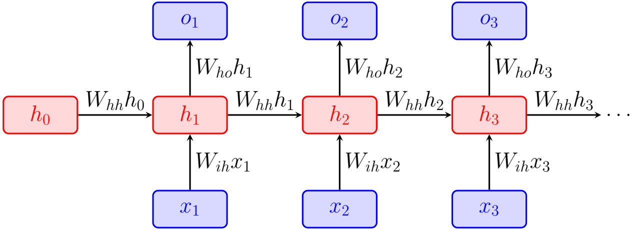
Recurrent neural networks

How much context is needed?

... I went ______ ______

Idea: combine new context with old context recurrently to handle varying context sizes

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



Backpropogation through time

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

[board]

Problem:

- Gradient involves repeated multiplication of W_{hh}
- Gradient will vanish / explode

Quick fixes:

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

Long-short term memory (LSTM)

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

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

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

- Input gate and forget gate

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

- Hidden state

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

Gating allows the network to learn to control how much gradient should vanish.

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:

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

- Base of log and exponentiation should match
- Exponent is cross entropy: $H(p_{data}, p_\theta) = -\mathbb{E}_{x\sim p} \log p_\theta(x)$.
- Interpretation: a model of perplexity k predicts the next word by throwing a fair k-sided die.