

4. Language Modeling with n-grams and RNNs

Language Modeling

- Alphabet Σ is a finite, non-empty set of symbols
- A string over an alphabet Σ is any finite sequence of alphabet symbols.
- The set of all possible strings is Σ^* , the Kleene closure of Σ (output Y).

Example: $\Sigma = \{a, b, c\}$, $Y_1 = a$, $Y_2 = aab$, $Y_3 = abcb \dots a$, $\dots$

$$\Sigma^* = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaaa, aacb, aac, \dots\}$$

↑ ↑
infinite empty string

Formally, a language model is a distribution over Σ^* , it assigns probabilities to strings $Y \in \Sigma^*$.

Globally Normalized Language Models

Given an arbitrary scoring function $\text{score}: Y \mapsto \mathbb{R}$, we define

$$p(Y) = \frac{\exp[\text{score}(Y)]}{\sum_{Y' \in \Sigma^*} \exp[\text{score}(Y')]} = \frac{1}{Z} \exp[\text{score}(Y)]$$

normalization constant
an infinite sum, NOT always computable.

Locally Normalized Language Models

A locally normalized (autoregressive) language model is a collection of conditional probability distributions $p(Y_i | Y)$, meaning given the generated string $Y \in \Sigma^*$, how probable is $Y_i \in \Sigma \cup \{\text{EOS}\}$ to follow?

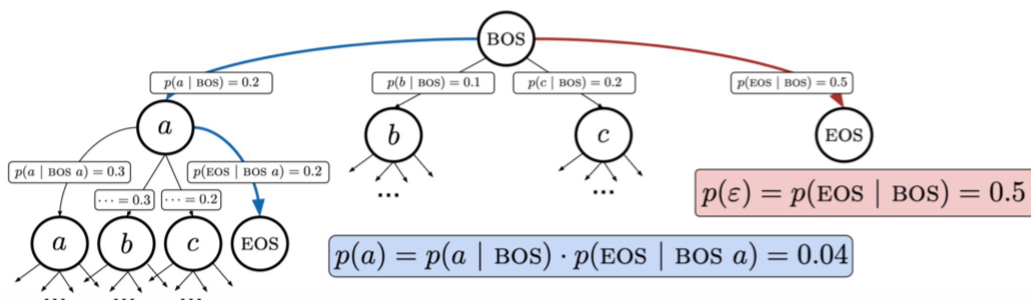
- Beginning-of-sequence BOS $\rightarrow p(Y | \text{BOS})$
- End-of-sequence EOS $\rightarrow p(\text{EOS} | Y)$

Example: Compute String probability

$$Y = Y_1 Y_2 \dots Y_N, \quad Y_{<N} = Y_1 Y_2 \dots Y_{N-1}$$

$$p(Y) = p(Y_1 | \text{BOS}) p(Y_2 | \text{BOS } Y_1) \dots p(Y_N | Y_{<N}) p(\text{EOS} | Y)$$

Visualizing as a prefix tree:



- * Local Normalization guarantees that the normalization constant is 1.
- * The sum of the probability of all children given their parent is 1.
- * Every node has an EOS as a descendant.

Tightness

- A locally normalized language model that sums to 1 is called tight.
- A non-tight loses probability to infinitely long structures. → Sequence model
- To ensure a model is tight, we force $p(\text{EOS} | \text{parent}) > \epsilon > 0$ for every parent node for some constant ϵ .

n-gram Language Modeling

The n-gram assumption: limit the context to the previous n-1 symbols.

$$P(Y_t | Y_{<t}) = P(Y_t | Y_{t-n+1} \dots Y_{t-1})$$

* n-grams ensure a finite number of histories.

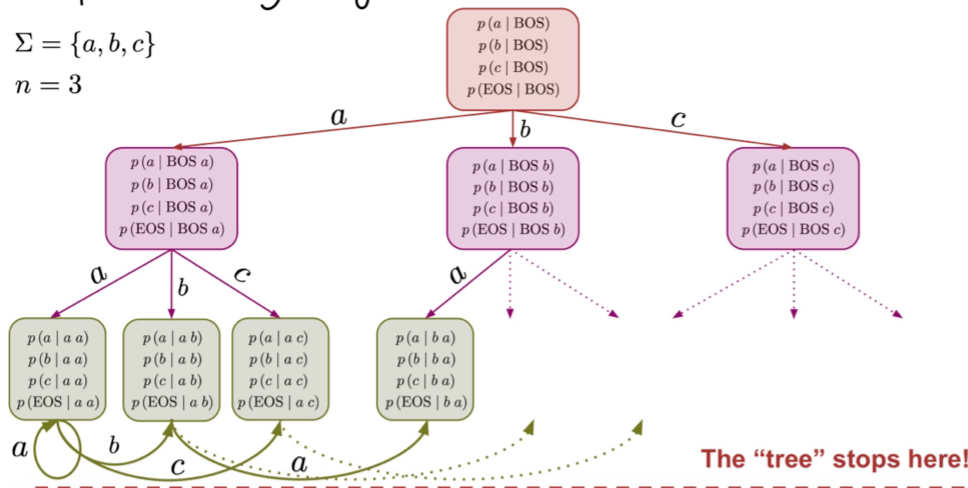
Probability of the entire string $Y \in \Sigma^*$ is:

$$P(Y) = P(Y_1 \dots Y_T) = P(\text{EOS} | Y_{T-n+2} \dots Y_T) \prod_{t=1}^T P(Y_t | Y_{t-n+1} \dots Y_{t-1})$$

$$Y_0 = Y_{-1} \dots = Y_{-n+2} = \text{BOS}$$

Example: Estimating n-grams model visualization

$\Sigma = \{a, b, c\}$
 $n = 3$

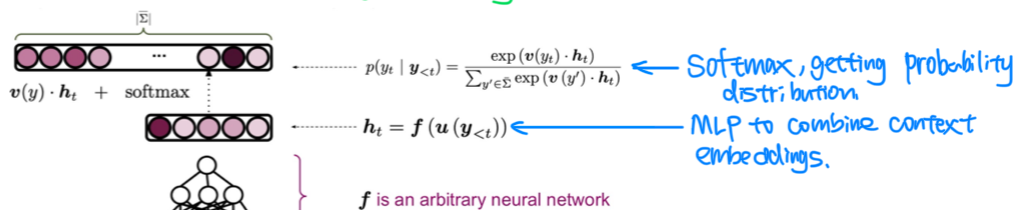


Better than naive n-grams model - intuition for RNN

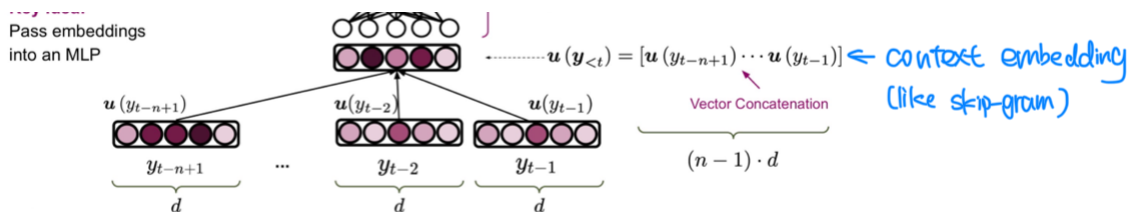
$$P(Y_t | Y_{<t}) = \frac{\exp(v(Y_t) \cdot h_t)}{\sum_{y' \in \Sigma} \exp(v(y') \cdot h_t)}$$

- $v(Y_t)$: word embeddings, representing individual symbols.
- h_t : context embeddings, summarize strings of n-1 symbols

* Word embeddings are like shared parameters used by MLP, reducing the number of parameters and generalizing the model.

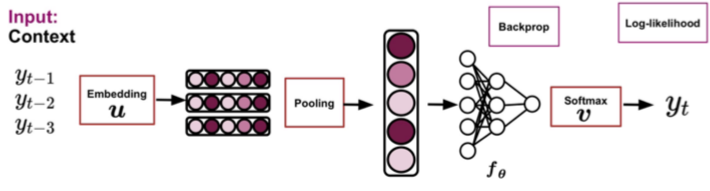


Key idea:



← context embedding (like skip-gram)

Whole pipeline

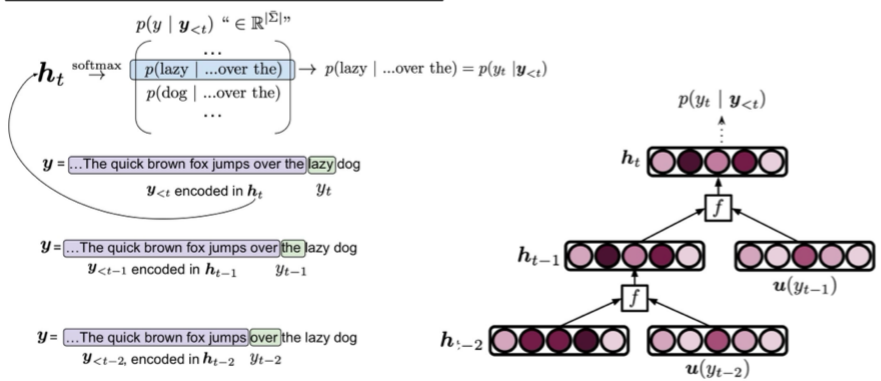


$$p(y^{(u)}) = \prod_{t=1}^{|y^{(u)}|} p(y_t^{(u)} | y_{t-1}^{(u)}, \dots, y_{t-n+1}^{(u)})$$

$$\sum_{y^{(u)} \in \mathcal{D}} \log p(y^{(u)}) = \sum_{y^{(u)} \in \mathcal{D}} \sum_{t=1}^{|y^{(u)}|} \log p(y_t^{(u)} | y_{t-1}^{(u)}, \dots, y_{t-n+1}^{(u)})$$

$$= \sum_{y^{(u)} \in \mathcal{D}} \sum_{t=1}^{|y^{(u)}|} (w_{x_0} \cdot h_t - \log(\sum_{y \in \mathcal{V}} \exp(w_y \cdot h_t)))$$

Recurrent Neural Networks

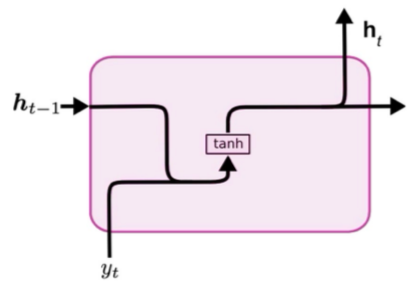


f defines the type of the RNN, describes how we combine the recurrent part of the model with the representation of the input at current time step.

Vanilla RNNs

(Vanilla) Recurrent Neural Networks

- There are many ways of framing an RNN, but at its core it is just a non-linear combination of the recurrent state and the inputs.



Elman recurrence (Elman, 1990):

$$h_t = \sigma(Uh_{t-1} + Vu(y_{t-1}) + b_h)$$

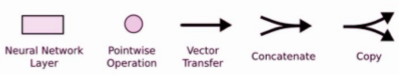
Variant:

$$h_t = \sigma(W[h_{t-1}; u(y_{t-1})])$$

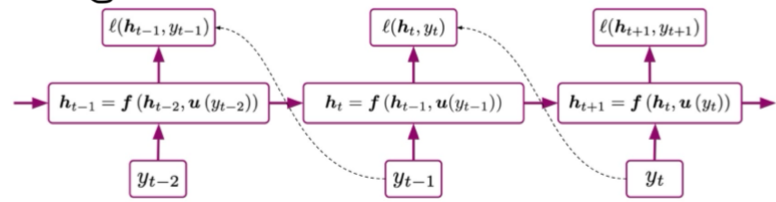
where $W \in \mathbb{R}^{d \times 2d}$, and $U, V \in \mathbb{R}^{d \times d}$ are the recurrence matrices, σ is a non-linearity (e.g., tanh) as in an MLP

Note: There should be a base condition h_0 , which is a trainable parameter

from Christopher Olah's blog, <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>



Training RNNs



- Learn U, V, b, u, v
- Trained using backpropagation through time (temporal hidden-state dependencies).

* First unroll the recurrent neural network and do backpropagation. For any input sentence, the unrolled network is finite and can be propagated through.

