Neural Language Models

Kyunghyun Cho
New York University
Courant Institute (Computer Science) and Center for Data Science
Facebook AI Research

Language Modelling

- Input: a sentence
- Output: the probability of the input sentence
- A language model captures the distribution over all possible sentences.
  $p(X) = p((x_1, x_2, \ldots, x_T))$
- Unlike text classification, it is *unsupervised learning*.
  - We will however turn the problem into a *sequence of supervised learning*.

Autoregressive language modelling

- Autoregressive sequence modelling
  - The distribution over the next token is based on all the previous tokens.
    $p(X) = p(x_1)p(x_2|x_1) \cdots p(x_T|x_1, \ldots, x_{T-1})$
  - This equality holds exactly due to the def. of conditional distribution*
- Unsupervised learning becomes a set of supervised problems.
  - Each conditional is a neural network classifier.
  - Input is all the previous tokens (a partial sentence).
  - Output is the distribution over all possible next tokens (classes).
  - It is a **text classification** problem.

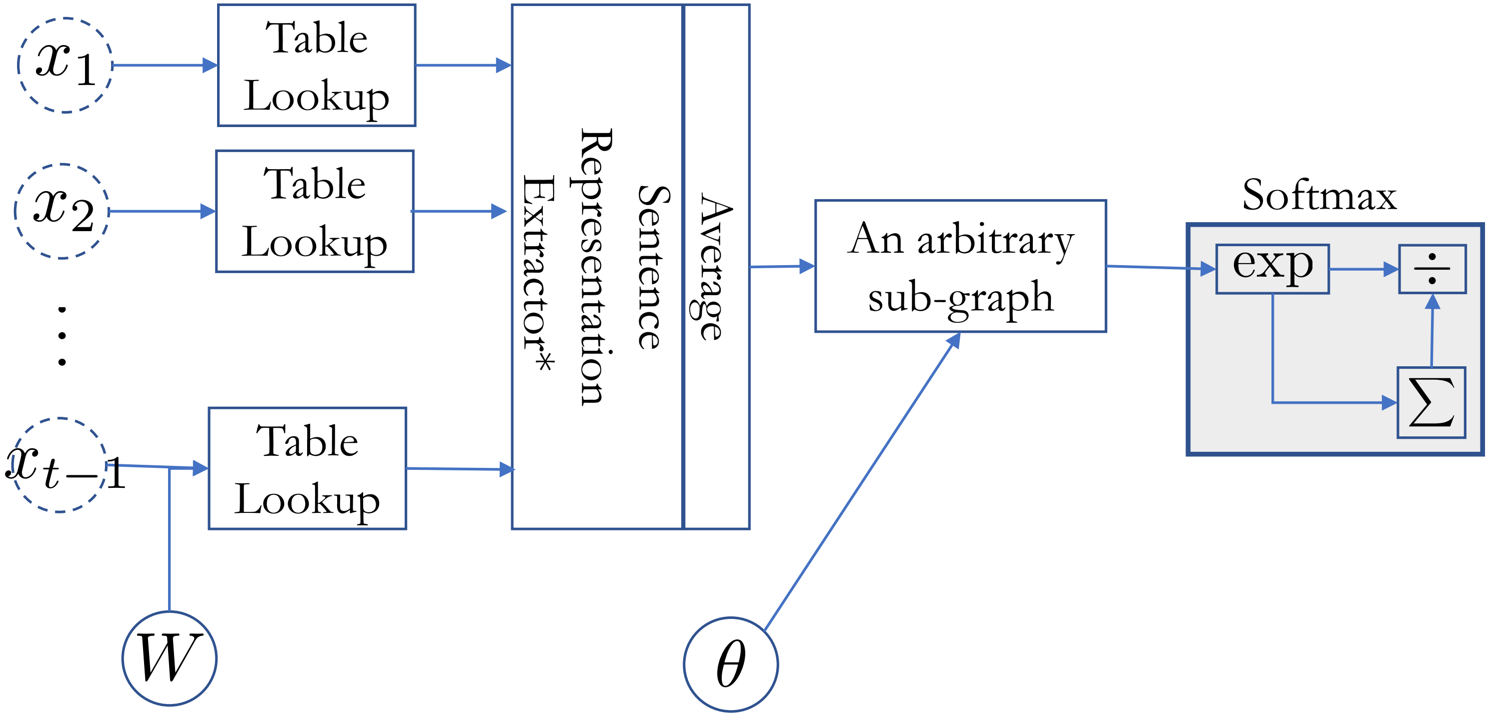
* See Lecture 1.

Autoregressive language modelling

- Autoregressive sequence modelling
  - The distribution over the next token is based on all the previous tokens.

$$p(X) = p(x_1)p(x_2|x_1)\cdots p(x_T|x_1,\ldots,x_{T-1})$$

- Each conditional is a sentence classifier:

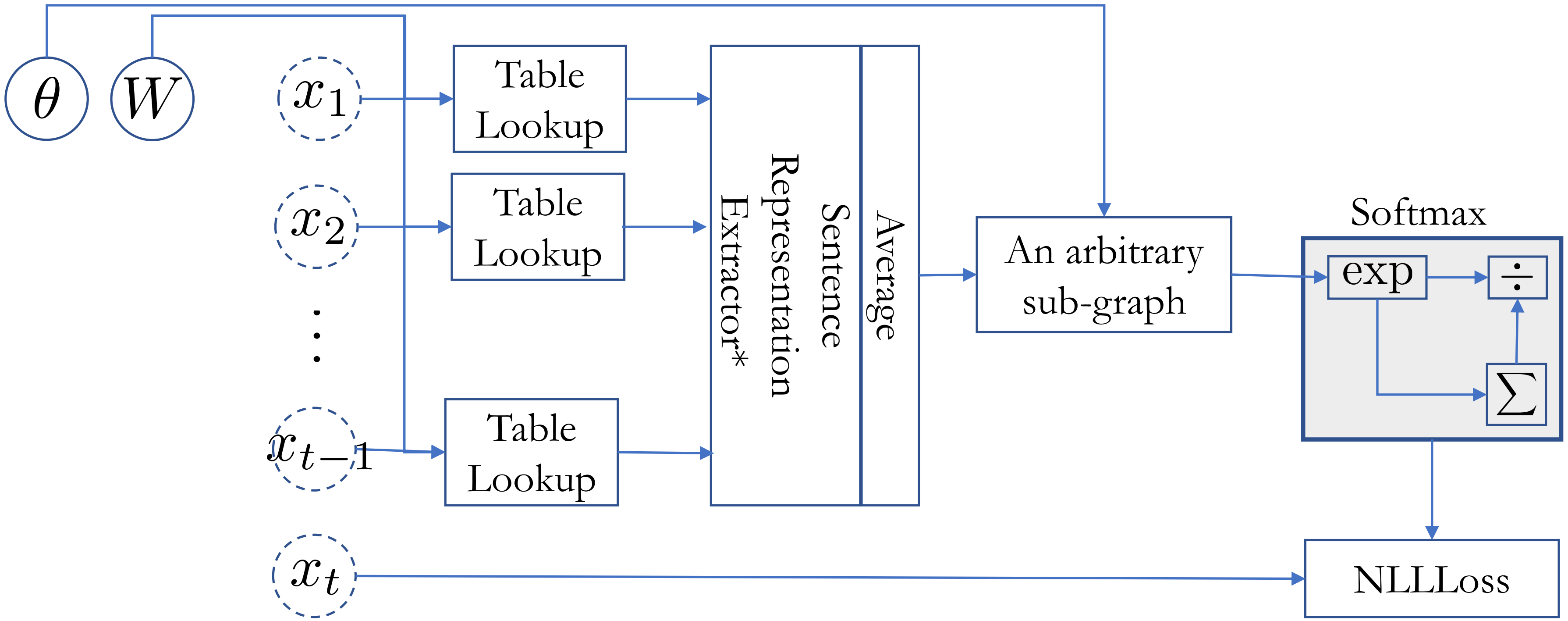


* See Lecture 2.

Autoregressive language modelling

- Autoregressive sequence modelling $p(X) = \prod_{t=1}^{T} p(x_t | x_{<t})$
- Loss function: the sum of negative log-probabilities

$$\log p_\theta(X) = \sum_{n=1}^{N} \sum_{t=1}^{T} \log p_\theta(x_t | x_{<t})$$



Scoring a sentence

- Autoregressive sequence modelling
  - The distribution over the next token is based on all the previous tokens.
    $p(X) = p(x_1)p(x_2|x_1)\cdots p(x_T|x_1,\ldots,x_{T-1})$
- A natural way to score a sentence:
  - In Korea, more than half of residents speak Korean.
  - "In" is a reasonable token to start a sentence.
  - "Korea" is pretty likely given "In"
  - "more" is okay token to follow "In Korea"
  - "than" is very likely after "In Korea, more"
  - "half" is also very likely after "In Korea, more than"

$$\vdots$$

- Sum all these scores and get the sentence score.

Scoring a sentence

- Autoregressive sequence modelling
  - The distribution over the next token is based on all the previous tokens.
    $p(X) = p(x_1)p(x_2|x_1) \cdots p(x_T|x_1, \ldots, x_{T-1})$
- A natural way to score a sentence:
  - "In Korea, more than half of residents speak Korean."
    vs.
    "In Korea, more than half of residents speak Finnish."
  - The former is more likely (=higher probability) than the latter.
- This is precisely what NLLLoss computes over the sentence.

N-Gram Language Models

- Let's back up a little…
- What would we do *without* a neural network?
- We need to estimate *n*-gram probabilities: $p(x|x_{-N}, x_{-N+1}, \ldots, x_{-1})$
- Recall the def. of conditional and marginal probabilities:

$$p(x|x_{-N}, x_{-N+1}, \ldots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{p(x_{-N}, x_{-N+1}, \ldots, x_{-1})}$$

$$= \frac{p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{\sum_{x \in V} p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}$$

  - V : all possible tokens (=vocabulary)

N-Gram Language Models

- We need to estimate *n*-gram probabilities:

$$p(x|x_{-N}, x_{-N+1}, \dots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}{\sum_{x \in V} p(x_{-N}, x_{-N+1}, \dots, x_{-1}, x)}$$

- How do we estimate the probability?
  - I want to estimate the probability of my distorted coin landing head.
  - **M**aximum **l**ikelihood **e**stimation (MLE):
    toss the coin a lot and look at how often it lands heads.

Data Collection　　　　Estimation

N-Gram Language Models

- We need to estimate n-gram probabilities:

$$p(x|x_{-N}, x_{-N+1}, \ldots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{p(x_{-N}, x_{-N+1}, \ldots, x_{-1})}$$

- Data: all the documents or sentences you can collect
    - e.g., Wikipedia, news articles, tweets, …
- Estimation:
    1. Count the # of occurrences for the n-gram $(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)$
    2. Count the #'s of occurrences for all the n-grams of the form: $(x_{-N}, x_{-N+1}, \ldots, x_{-1}, ?)$

N-Gram Language Models

- We need to estimate *n*-gram probabilities:

$$p(x|x_{-N}, x_{-N+1}, \ldots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{p(x_{-N}, x_{-N+1}, \ldots, x_{-1})}$$

- Estimation:

$$\begin{aligned} p(x|x_{-N}, x_{-N+1}, \ldots, x_{-1}) &= \frac{p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{\sum_{x \in V} p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)} \\ &\approx \frac{c(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{\sum_{x' \in V} c(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x')} \end{aligned}$$

- *Do you see why this makes sense?*

N-Gram Language Models

- We need to estimate n-gram probabilities:

$$p(x|x_{-N}, x_{-N+1}, \ldots, x_{-1}) = \frac{p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{\sum_{x \in V} p(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}$$

$$\approx \frac{c(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x)}{\sum_{x' \in V} c(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x')}$$

- How likely is “University” given “New York”?
  - Count all “New York University”
  - Count all “New York ?”: e.g., “New York State”, “New York City”, “New York Fire”, “New York Police”, “New York Bridges”, …
  - How often “New York University” happens among these?

N-Gram Language Models – Two problems

1. Data sparsity: lack of generalization
   - What happens "one" n-gram never happens?

$$p(\text{a lion is chasing a llama}) = p(\text{a}) \times p(\text{lion}|\text{a}) \times p(\text{is}|\text{a lion}) \times p(\text{chasing}|\text{lion is}) \times p(\text{a}|\text{is chasing}) \times \underbrace{p(\text{llama}|\text{chasing a})}_{=0} = 0$$

2. Inability to capture long-term dependencies
   - Each conditional only considers a small window of size *n*.
   - Consider "*the same **stump** which had impaled the car of many a guest in the past thirty years and which he refused to have **removed***"
   - It is impossible to tell "removed" is likely by looking at the four preceding tokens.

Traditional Solutions

1. Data Sparsity
    - Smoothing: add a small constant to avoid 0.

$$p(x|x_{-N}, x_{-N+1}, \ldots, x_{-1}) \approx \frac{c(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x) + \epsilon}{\epsilon|V| + \sum_{x' \in V} c(x_{-N}, x_{-N+1}, \ldots, x_{-1}, x')}$$

    - Backoff: try a shorter window.

$$c(x_{-N}, \ldots, x) = \begin{cases} \alpha c(x_{-N+1}, \ldots, x) + \beta, & \text{if } c(x_{-N}, \ldots, x) = 0 \\ c(x_{-N}, \ldots, x), & \text{otherwise} \end{cases}$$

    - The most widely used approach: Kneser-Ney smoothing/backoff
    - **KenLM** implements the efficient n-gram LM model.

Traditional Solutions

2. Long-Term Dependency
    - Increase *n*: not feasible as the data sparsity worsens.
    - # of all possible *n*-grams grows exponentially w.r.t. *n*: $O(|V|^n)$
    - The data size does not grow exponentially: many never-occurring *n*-grams.

- These two problems are closely related and cannot be tackled well.
    - To capture long-term dependencies, *n* must be large.
    - To address data sparsity, *n* must be small.
    - Conflicting goals..

N-Gram Language Models – Two problems

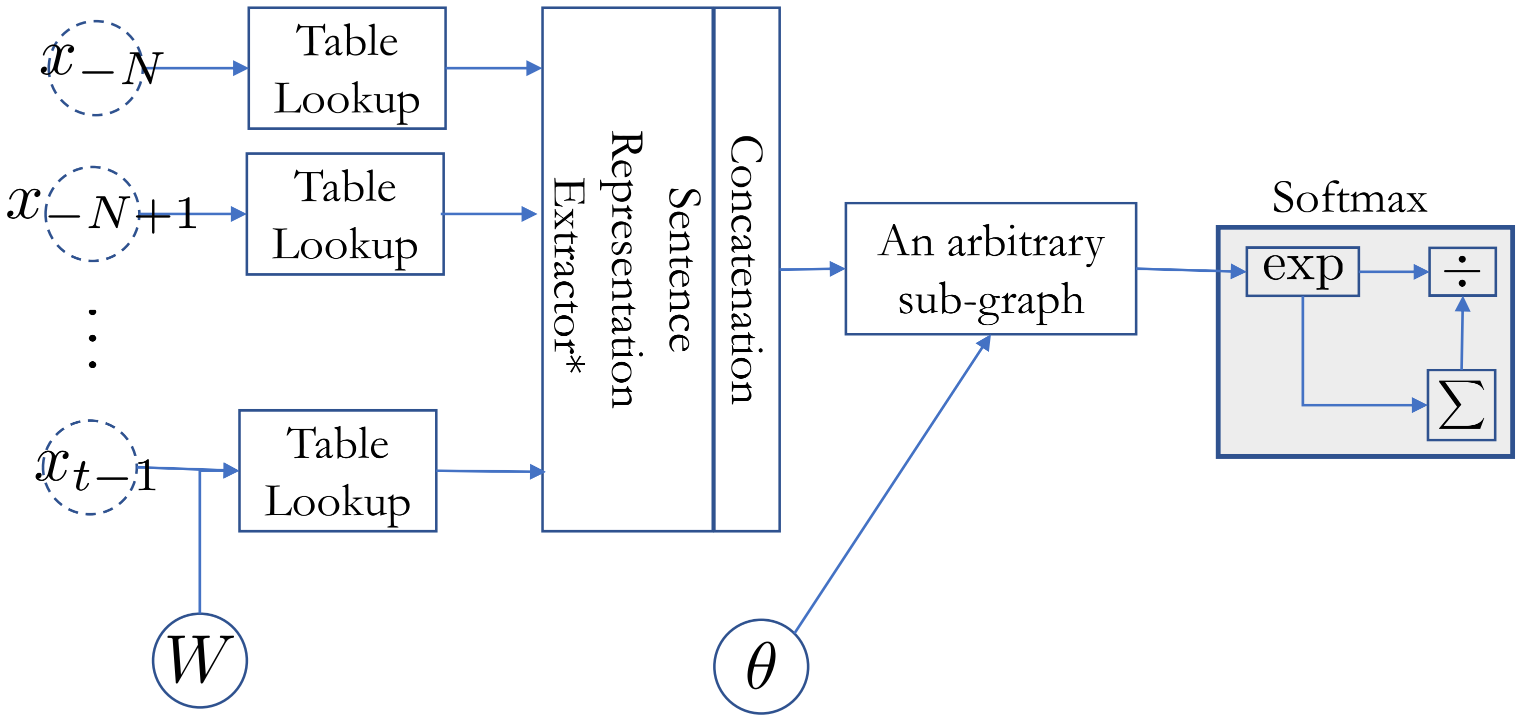
1. Data sparsity: lack of generalization
    - What happens "one" n-gram never happens?

$$p(\text{a lion is chasing a llama}) = p(\text{a}) \times p(\text{lion}|\text{a}) \times p(\text{is}|\text{a lion}) \times p(\text{chasing}|\text{lion is}) \times p(\text{a}|\text{is chasing}) \times \underbrace{p(\text{llama}|\text{chasing a})}_{=0} = 0$$

2. Inability to capture long-term dependencies
    - Each conditional only considers a small window of size n.
    - Consider "*the same* ***stump*** *which had impaled the car of many a guest in the past thirty years and which he refused to have* ***removed***"
    - It is impossible to tell "removed" is likely by looking at the four preceding tokens.

Neural N-Gram Language Model [Bengio et al., 2001]

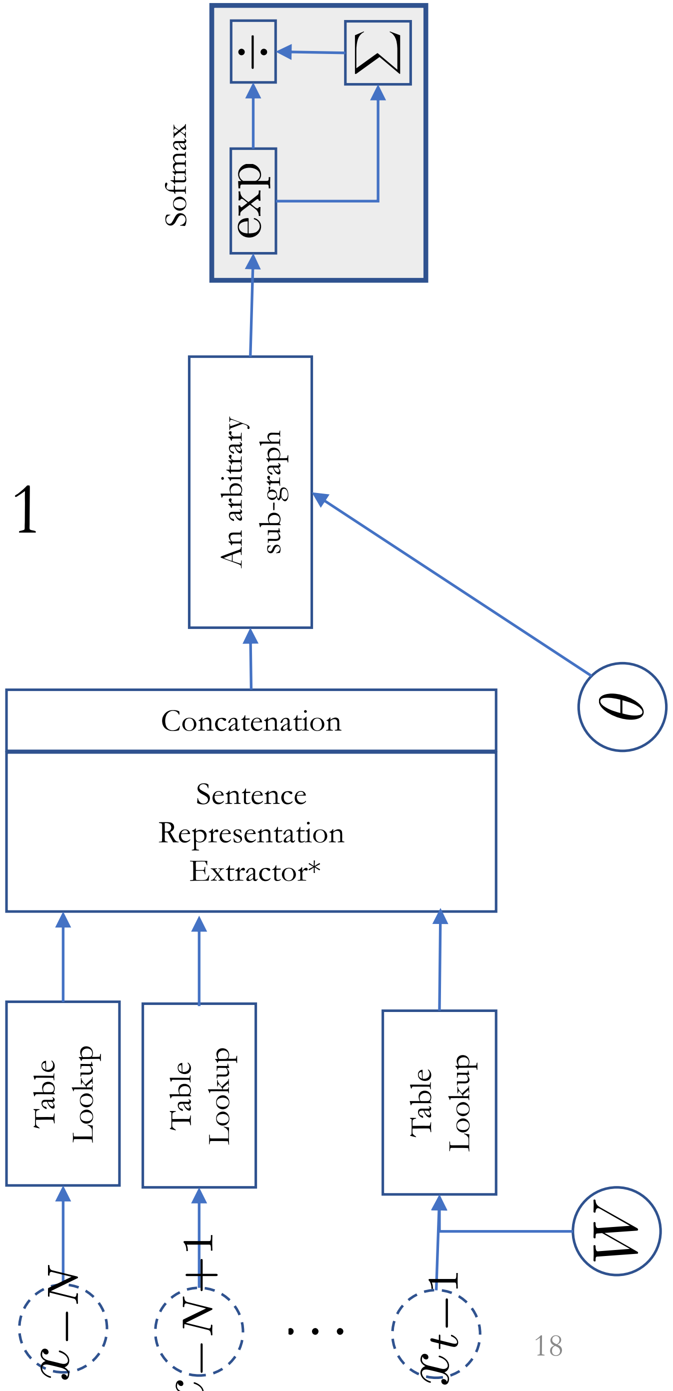
- The first extension of n-gram language models using a neural network



Neural N-Gram Language Model

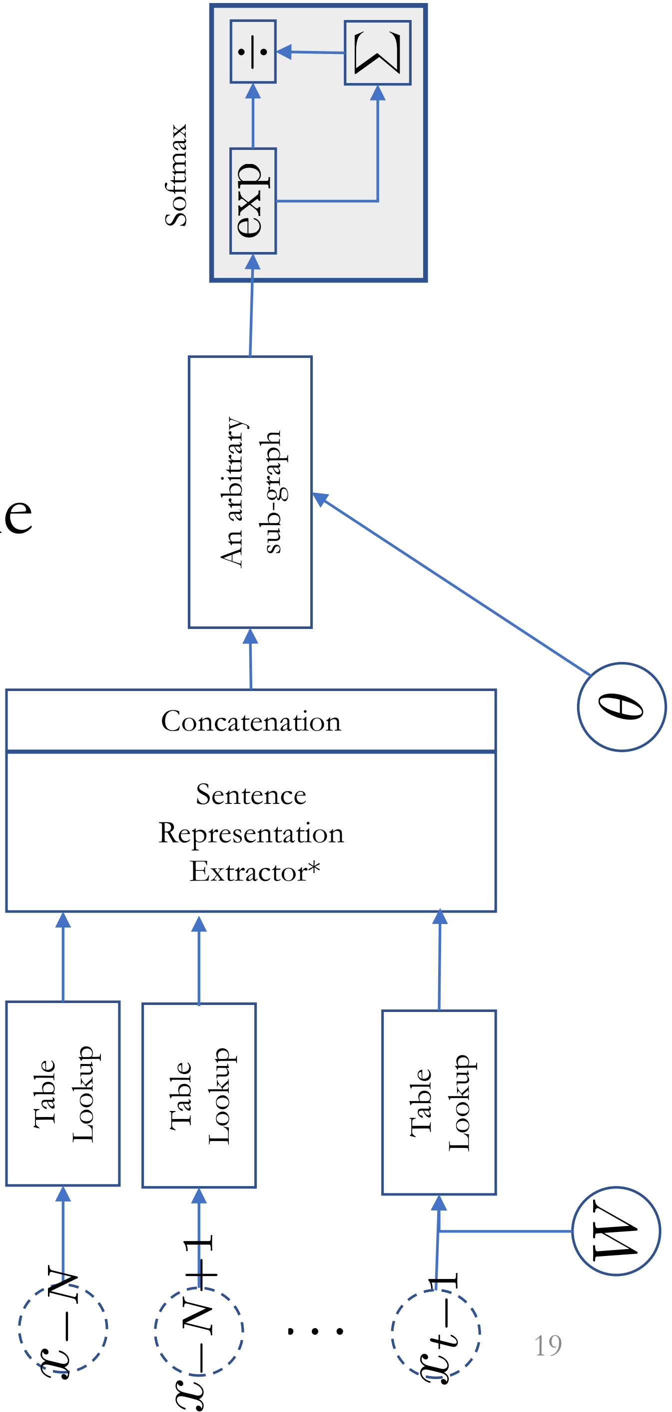
- The first neural language models
- Trained using backpropagation and SGD: see Lecture 1
- Generalizes to an unseen *n*-gram
- **Addresses the issue of data sparsity**

- *How?*



Neural N-Gram Language Model

- Why does the data sparsity happen?
- A "shallow" answer: some n-grams do not occur in the training data, while they do in the test time.
- A "slightly deeper" answer: it is difficult to impose token/phrase similarities in the discrete space.

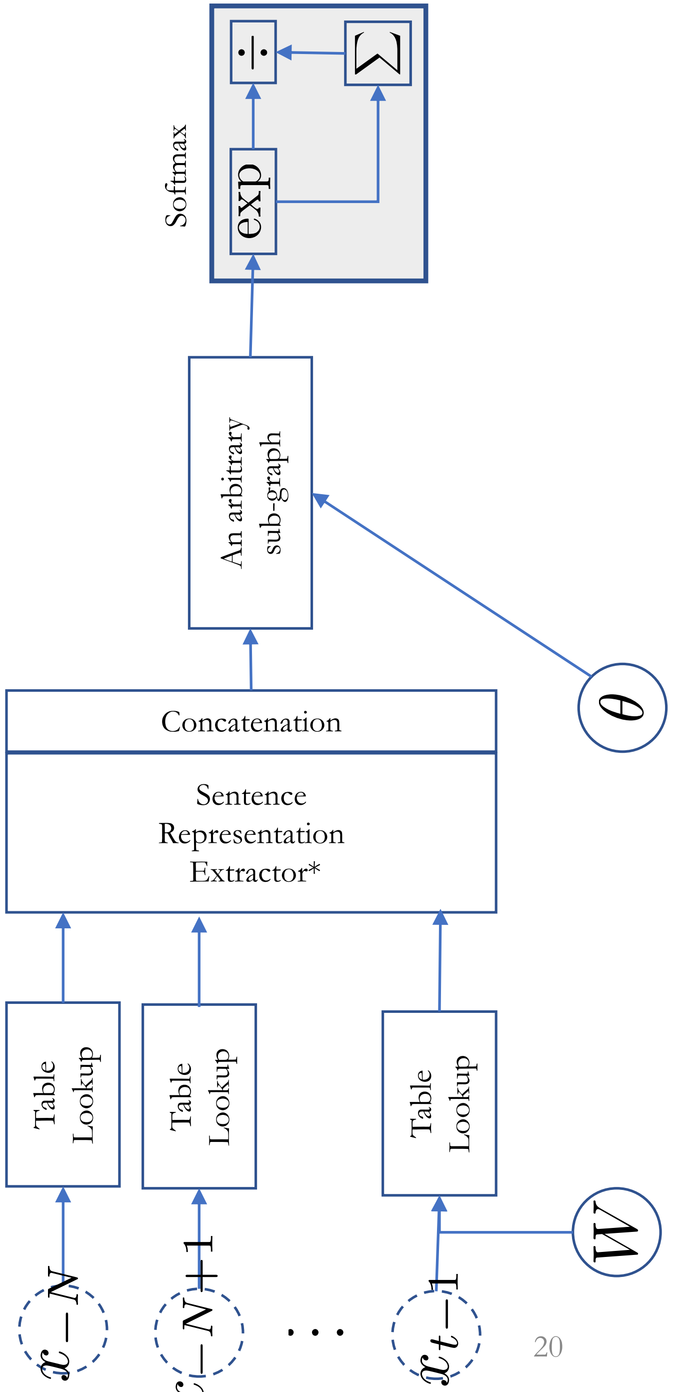


Neural N-Gram Language Model

- Why does the data sparsity happen?
- Back to the earlier example
  - Problem: $c(\text{chasing a llama}) = 0$
  - Observation: $c(\text{chasing a cat}) \gg 0$

$$c(\text{chasing a dog}) \gg 0$$
$$c(\text{chasing a deer}) \gg 0$$

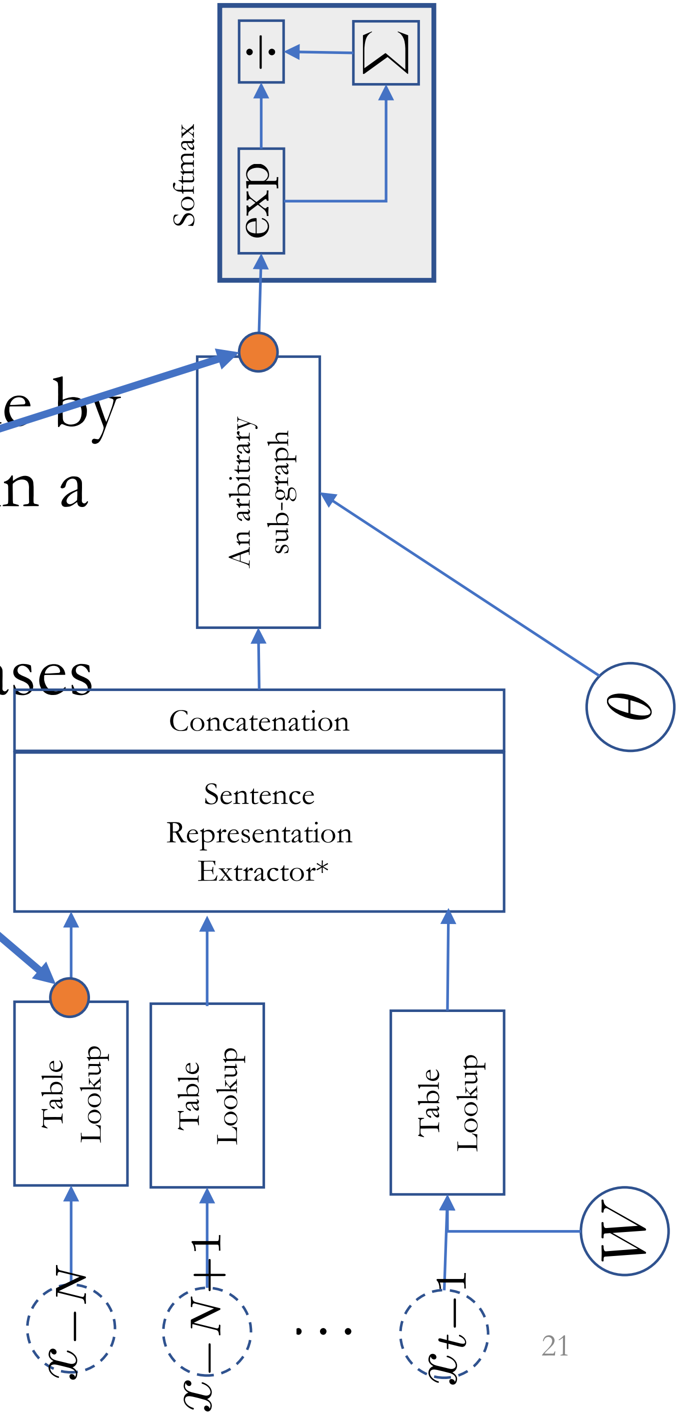
  - If the LM knew "llama" is a mammal similar to "cat", "dog" and "deer", it would be able to guess "chasing a llama" is as likely as "chasing a cat", "chasing a dog", and "chasing a deer".



Neural N-Gram Language Model

- The neural n-gram language model addresses this issue by "learning the similarities" among tokens and phrases in a "continuous vector space".
- In the "continuous vector space", similar tokens/phrases are nearby: e.g., word2vec [Mikolov et al., 2013; Pennington et al., 2014], doc2vec [Le&Mikolove, 2014], sentence-to-vec [Hill et al., 2016 and ref's therein]
- Then, similar input n-grams lead to similar output:

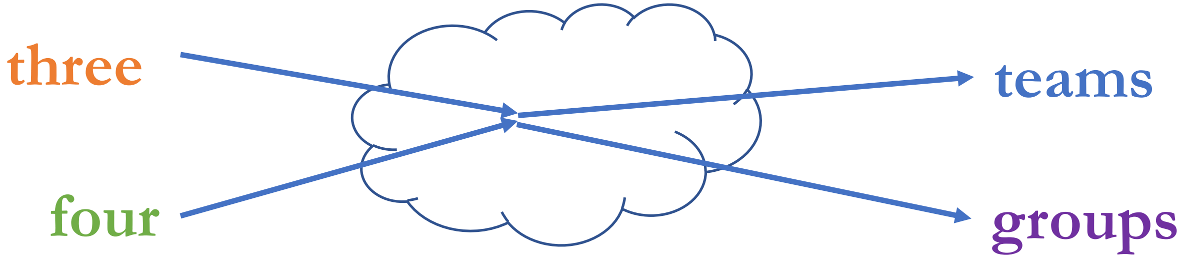
$$D(x_t|x_{t-N},\ldots,x_{t-1}\|x_t|x'_{t-N},\ldots,x'_{t-1}) < \epsilon$$

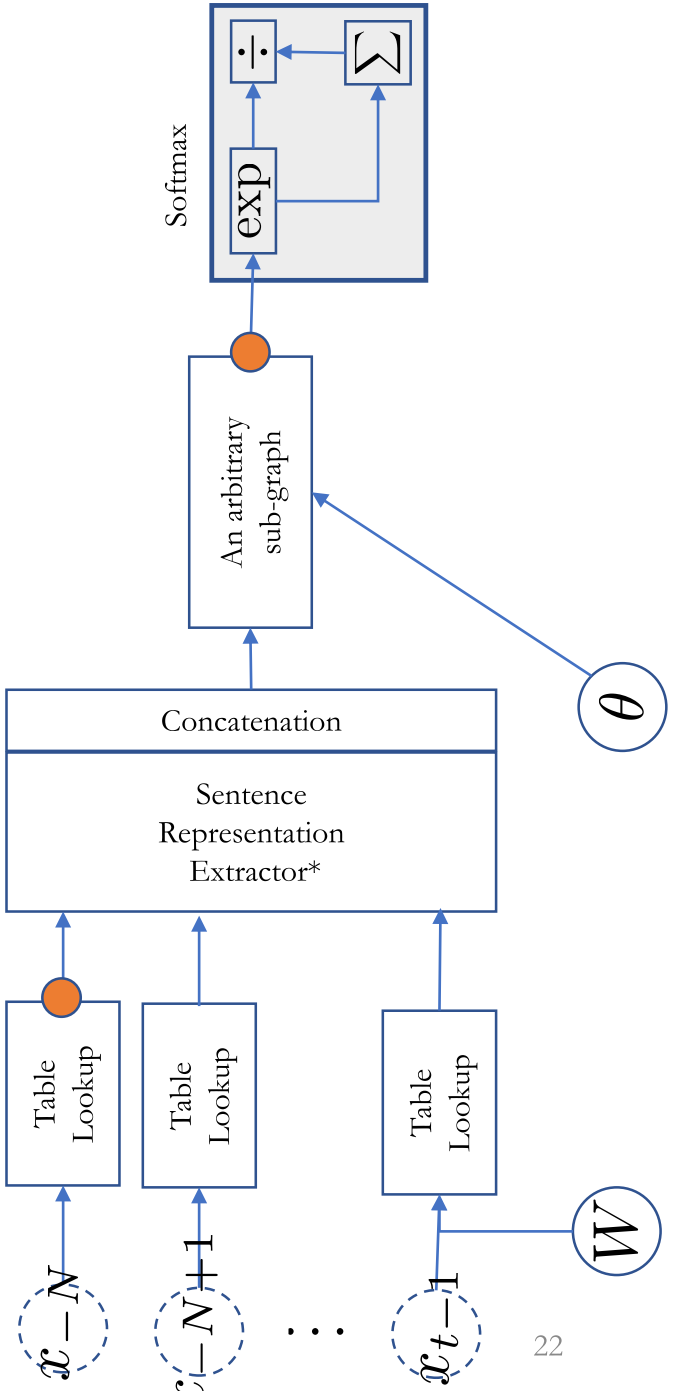


Neural N-Gram Language Model

- Training examples
  - there are three teams left for qualification.
  - four teams have passed the first round.
  - four groups are playing in the field.
- Q: how likely is “groups” followed by “three”?

Continuous vector space



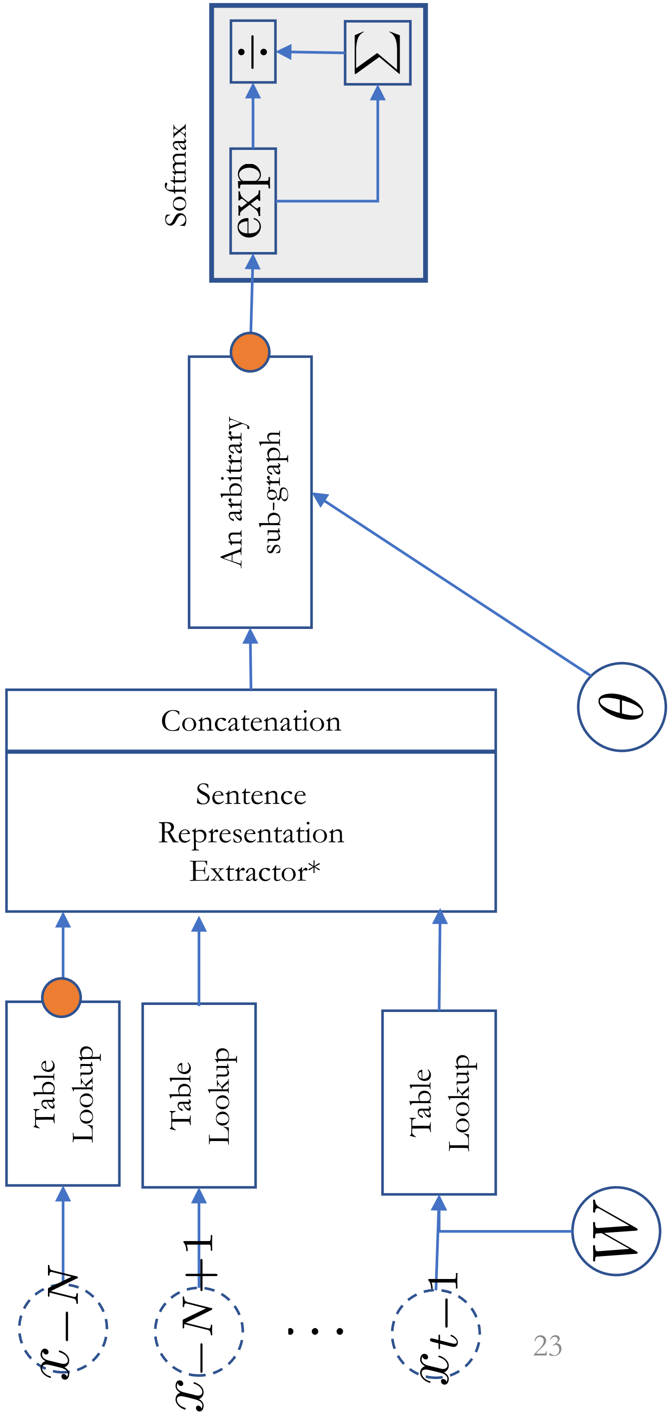


Neural N-Gram Language Model

- In practice,

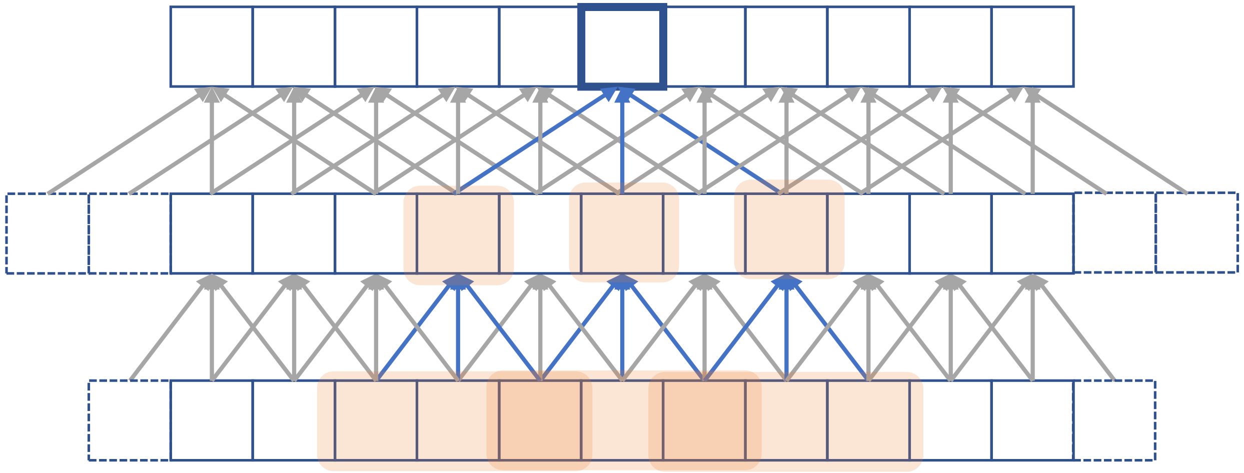
1. Collect all n-grams from the corpus.
2. Shuffle all the n-grams to build a training set
3. Train the neural n-gram language model using stochastic gradient descent on minibatches containing 100-1000 n-grams.
4. Early-stop based on the validation set.
5. Report perplexity on the test set.

$$\text{ppl} = b^{\frac{1}{|D|}\sum_{(x_1,\ldots,x_N)\in D}\log_b p(x_N|x_1,\ldots,x_{N-1})}$$



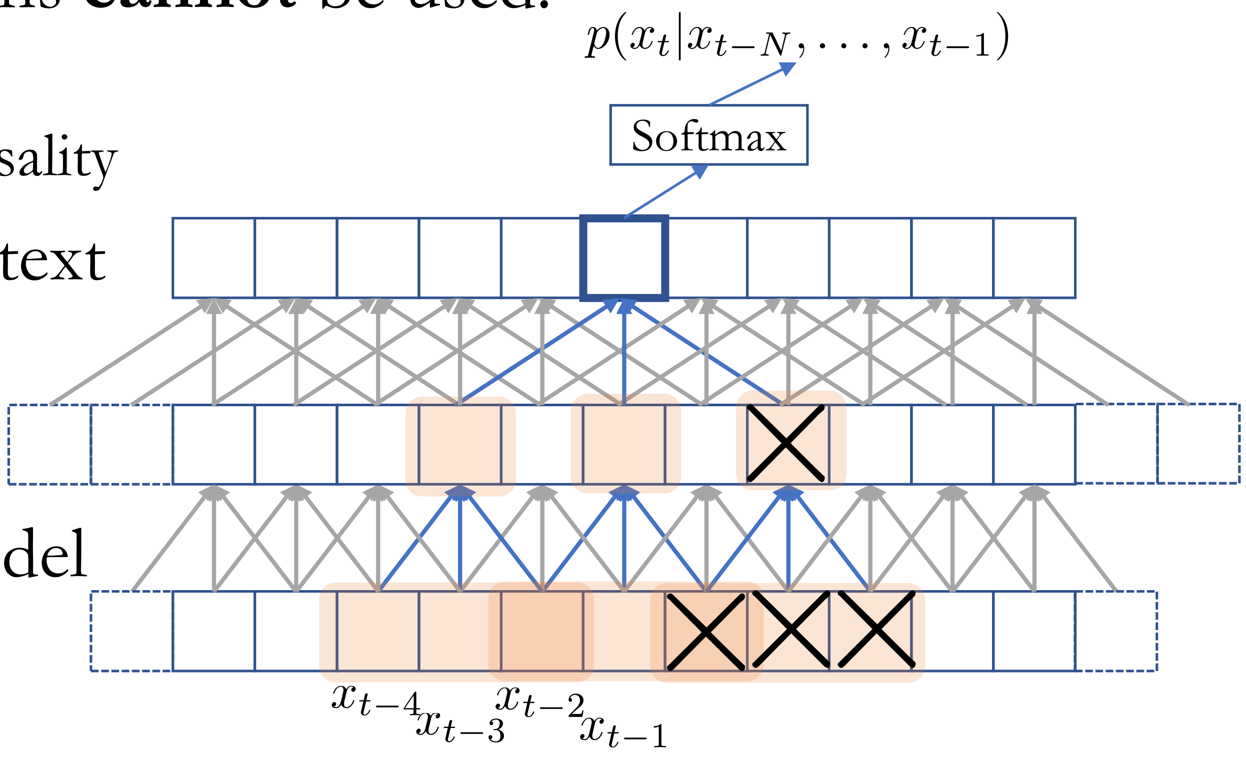
Increasing the context size – Convolutional Language Models [Kalchbrenner et al., 2015; Dauphin et al., 2016]

- Dilated convolution to rapidly increase the window size
  - Exponential-growth of the window by introducing a multiplicative factor
  - By carefully selecting the multiplicative factor, no loss in the information.



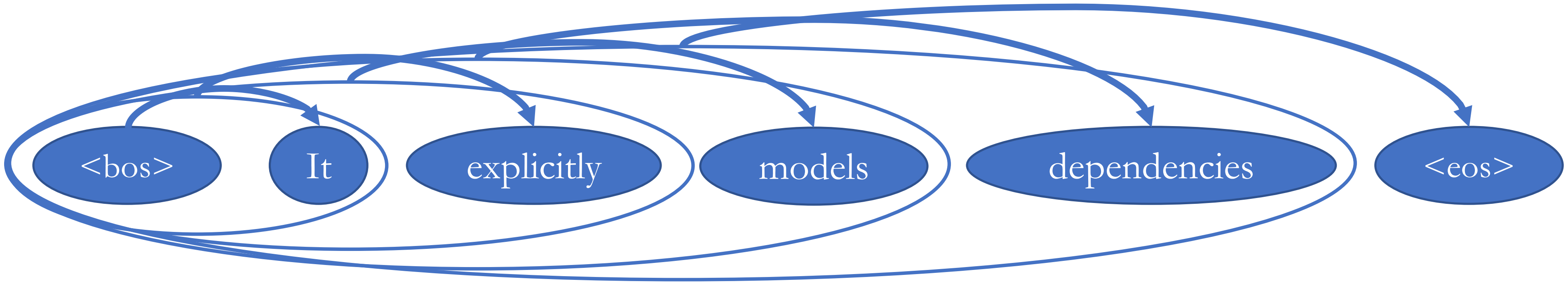
Increasing the context size – Convolutional Language Models [Kalchbrenner et al., 2015; Dauphin et al., 2016]

- Dilated convolution to rapidly increase the window size
- Causal convolution: the future tokens **cannot** be used.
  - Computation as usual: efficiency
  - Clever masking of future tokens: causality
- Efficient computation + larger context
- ByteNet [Kalchbrenner et al., 2015]
  - PixelCNN, WaveNet, …
- Gated Convolutional Language Model [Dauphin et al., 2016]



Causal sentence representation and language modelling

- Any sentence representation learning method from Lecture 2 could be used as long as it does not break the generative story:



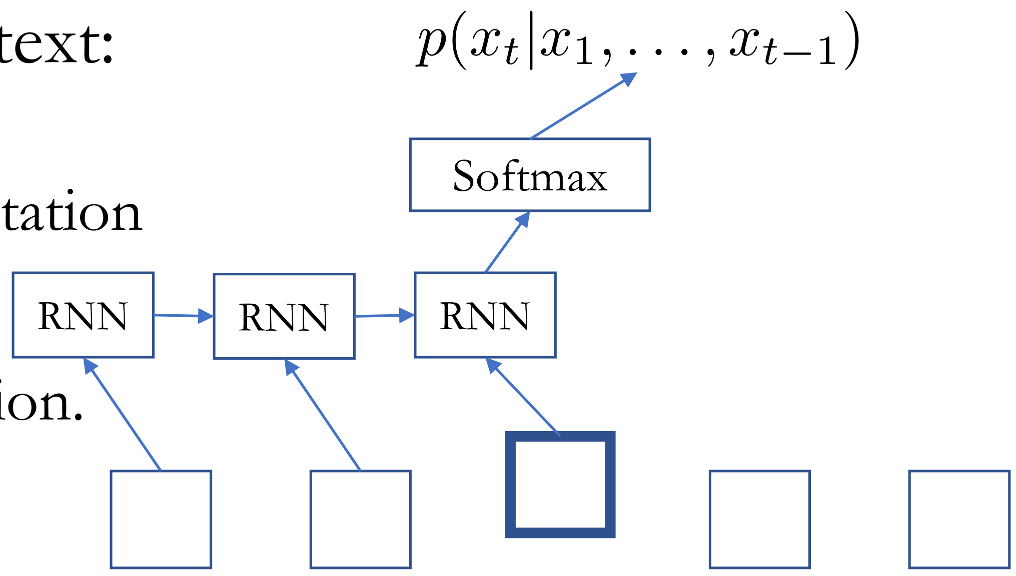
- In addition to the feedforward and convolutional n-gram language models, we can use any of the remaining sentence representation.

Infinite context $n \longrightarrow \infty$ – CBoW Language Models

- Equivalent to the neural LM after replacing "concat" with "average"
  - "Averaging" allows the model to consider the infinite large context window.
- Extremely efficient, but a weak language model
  - Ignores the order of the tokens in the context windows.
    - Any language with a fixed order cannot be modelled well.
  - Averaging ignores the absolute counts, which may be important:
    - If the context window is larger, "verb" becomes less likely in SVO languages.

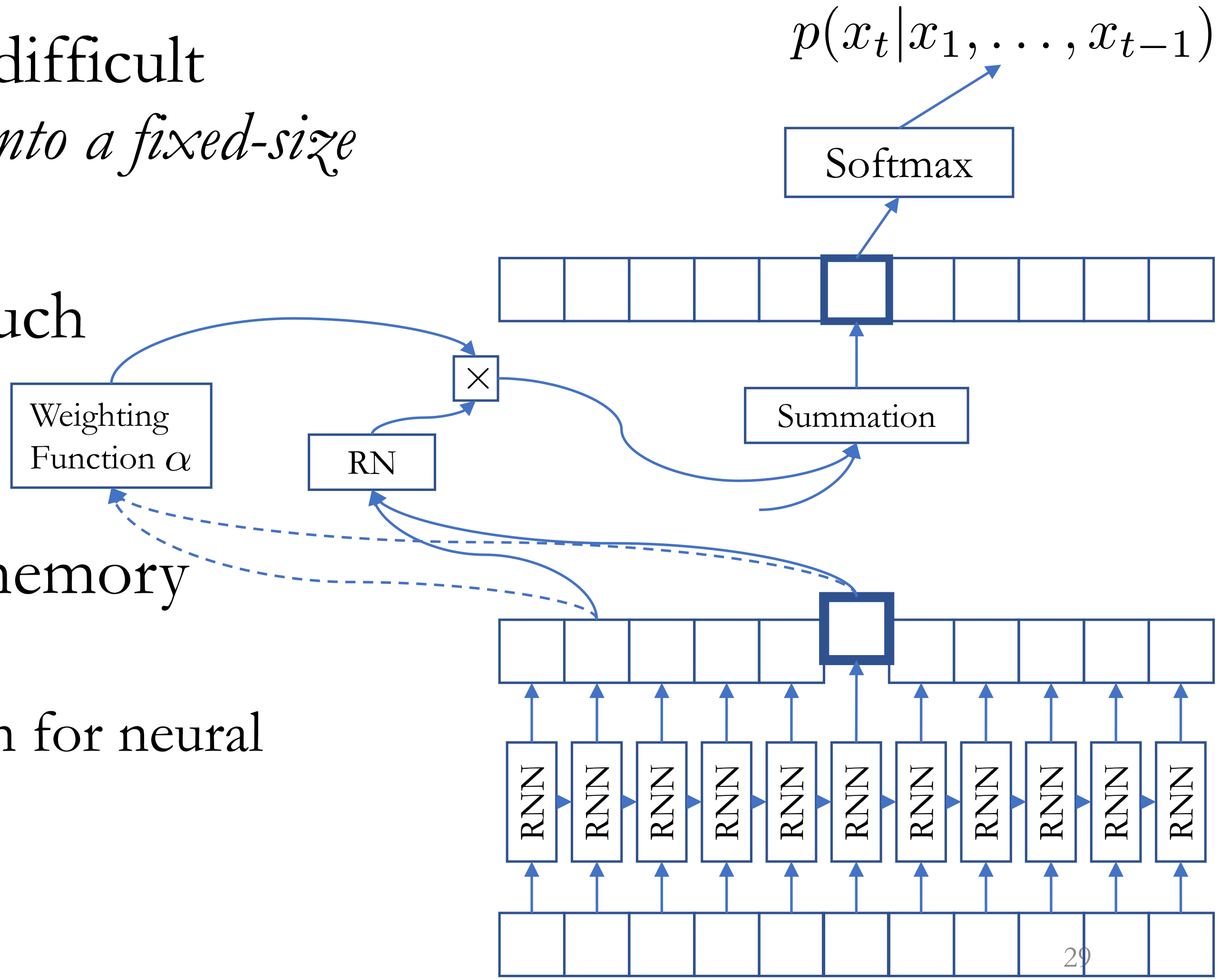
Infinite context n→∞ – Recurrent Language Models [Mikolov et al., 2010]

- A recurrent network summarizes all the tokens so far.
- Use the recurrent network's memory to predict the next token.
- Efficient online processing of a streaming text:
  - Constant time per step.
  - Constant memory throughout forward computation
- Useful in practice:
  - Useful for autocomplete and keyword suggestion.
  - Scoring partial hypotheses in generation.



Infinite context $n \rightarrow \infty$ – Recurrent Memory Networks [Tran et al., 2016]

- The **recurrent network** solves a difficult problem: *compress the entire context into a fixed-size memory vector.*
- **Self-attention** does not require such compression but still can capture long-term dependencies.
- Combine these two: a recurrent memory network (RMN) [Tran et al., 2016]
  - RNMT+: a similar, recent extension for neural machine translation



In this lecture, we learned

- What autoregressive language modelling is:
  $$p(X) = p(x_1)p(x_2|x_1)\cdots p(x_T|x_1, \ldots, x_{T-1})$$
- How autoregressive language modelling transforms unsupervised learning into a series of supervised learning:
  - It is a series of predicting the next token given previous tokens.
- How neural language modelling improves upon n-gram language models:
  - Continuous vector space facilitates generalization to unseen n-grams.
  - Infinitely large context window
- How sentence representation extraction is used for language modelling:
  - Convolutional language models, recurrent language models and self-attention language models..

In the next lecture,

- Sequence-to-Sequence Learning: Neural Machine Translation
  [Sutskever et al., 2014; Cho et al., 2014; Kalchbrenner&Blunsom, 2013]