Recap: Supervised Machine Learning

You all know already, but it's important enough to warrant repetition.

Supervised Learning – Overview

- Given:
  1. $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
  2. $l(M(x), y) \geq 0$
  3. $\mathcal{H}_1, \ldots, \mathcal{H}_M$
  4. Optimization algorithm
- Supervised learning finds an appropriate algorithm/model automatically
  1. For each hypothesis set $\mathcal{H}_m$, find the best model:

$$\hat{M}_m = \arg \min_{M \in \mathcal{H}_m} \sum_{n=1}^{N} l(M(x_n), y_n)$$

     using the optimization algorithm.

Supervised Learning – Overview

- Given:
  1. $D = \{(x_1, y_1), \dots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
  2. $l(M(x), y) \geq 0$
  3. $\mathcal{H}_1, \dots, \mathcal{H}_M$
  4. Optimization algorithm
- Supervised learning finds an appropriate algorithm/model automatically
  1. [Training] For each hypothesis set $\mathcal{H}_m$, find the best model:

$$\hat{M}_m = \arg\min_{M \in \mathcal{H}_m} \sum_{n=1}^{N} l(M(x_n), y_n)$$

     using the optimization algorithm and the **training set**.

Supervised Learning – Overview

- Given:
  1. $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
  2. $l(M(x), y) \geq 0$
  3. $\mathcal{H}_1, \ldots, \mathcal{H}_M$
  4. Optimization algorithm
- Supervised learning finds an appropriate algorithm/model automatically
  2. [Model Selection]* Among the trained models, select the best one

$$\hat{M} = \arg\min_{M \in \{\mathcal{H}_1, \ldots, \mathcal{H}_M\}} \sum_{(x,y) \in D_{\text{val}}} l(M(x), y)$$

using the **validation set** loss.

Supervised Learning – Overview

- Given:
  1. $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
  2. $l(M(x), y) \geq 0$
  3. $\mathcal{H}_1, \ldots, \mathcal{H}_M$
  4. Optimization algorithm
- Supervised learning finds an appropriate algorithm/model automatically
  3. [Reporting] Report how well the best model *would* work

$$R(\hat{M}) \approx \frac{1}{|D_{\text{test}}|} \sum_{(x,y) \in D_{\text{test}}} l(\hat{M}(x), y)$$

using the **test set** loss.

Supervised Learning – Overview

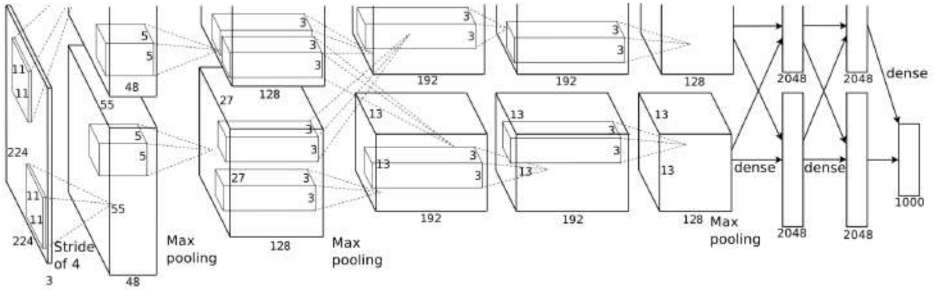
- Given:
    1. $D = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and $D_{\text{val}}, D_{\text{test}}$
    2. $l(M(x), y) \geq 0$
    3. $\mathcal{H}_1, \ldots, \mathcal{H}_M$
    4. Optimization algorithm
- Supervised learning finds an appropriate algorithm/model automatically
- It results in an algorithm $\hat{M}$ with an expected performance of $R(\hat{M})$.

Supervised Learning

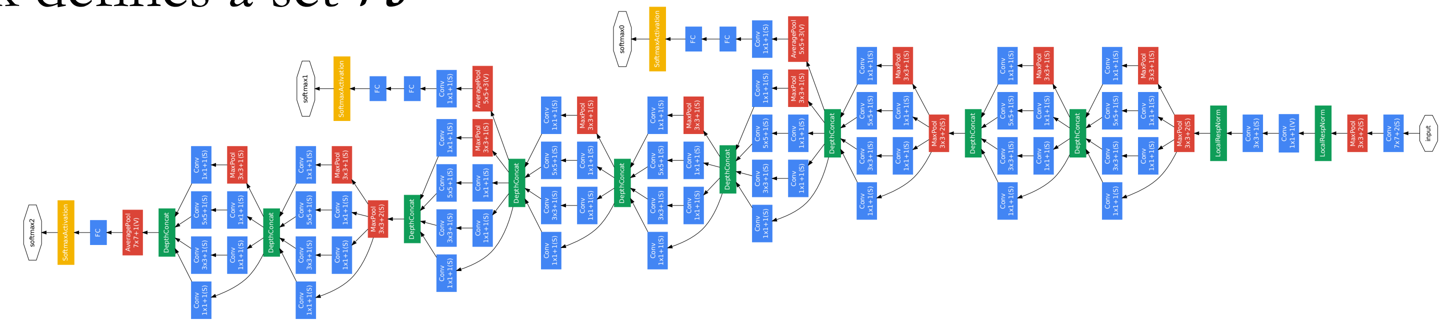
- Three points to consider both in research and in practice
  1. How do we decide/design a **hypothesis set**?
  2. How do we decide a **loss function**?
  3. How do we **optimize** the loss function?

Hypothesis set – Neural Networks

- In the case of deep learning,
  1. The architecture of a network defines a set $\mathcal{H}$



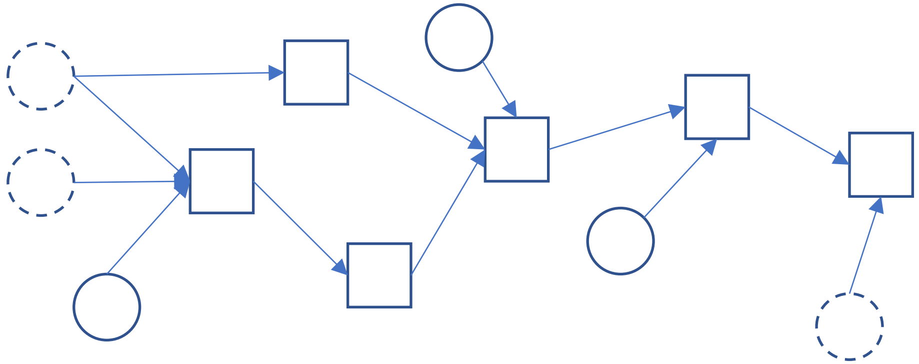
vs.



  2. Each model in the set $M \in \mathcal{H}$ is characterized by its parameters θ
     - Weights and bias vectors define one model in the hypothesis set.
- There are infinitely many models in a hypothesis set.
- We use optimization to find “a” good model from the hypothesis set.

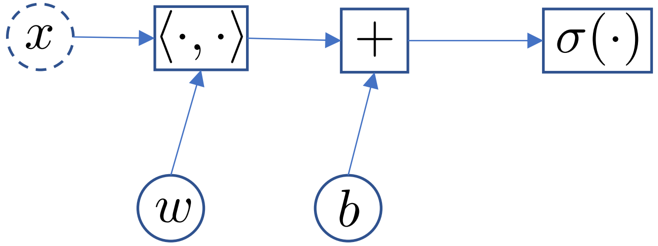
Network Architectures

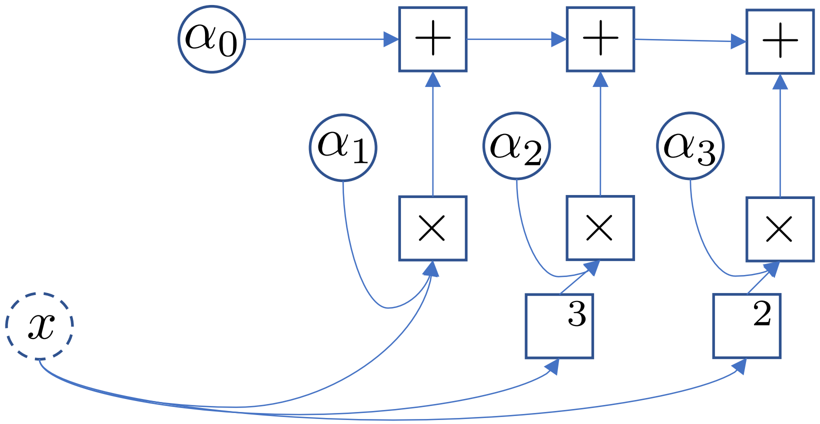
- What is a neural network? – An (arbitrary) directed acyclic graph (DAG)



1. Solid Circles ○ : parameters (to be estimated or found)
2. Dashed Circles ◌ : vector inputs/outputs (given as a training example)
3. Squares □ : compute nodes (functions, often continuous/differentiable)

Network Architectures

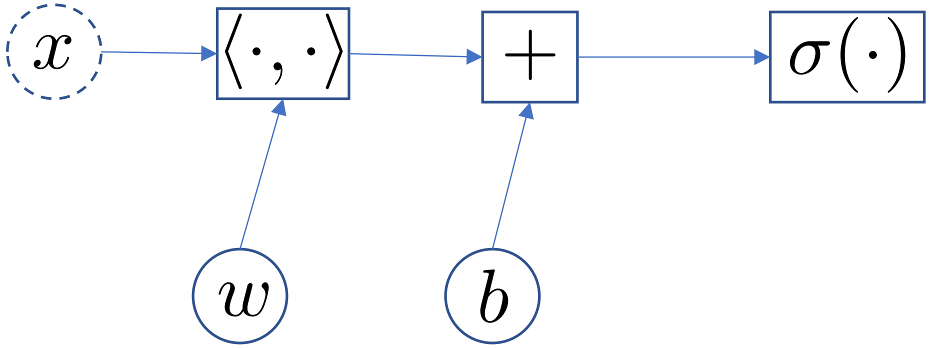
- What is a neural network? – An (arbitrary) directed acyclic graph (DAG)
  1. Logistic regression
     $$p_\theta(y=1|x) = \sigma(w^\top x + b) = \frac{1}{1+\exp(-w^\top x - b)}$$
     

  2. 3rd-order polynomial function $y = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3$
     

Inference – Forward Computation

- What is a neural network? – An (arbitrary) directed acyclic graph (DAG)
- Forward computation: how you "use" a trained neural network.
  - Topological sweep (breadth-first)
  - Logistic regression

$$p_\theta(y = 1|x) = \sigma(w^\top x + b) = \frac{1}{1 + \exp(-w^\top x - b)}$$



DAG $\leftrightarrow$ Hypothesis Set

- What is a neural network? – An (arbitrary) directed acyclic graph (DAG)
- Implication in practice
  - Naturally supports high-level abstraction
  - Object-oriented paradigm fits well.*
    - Base classes: variable (input/output) node, operation node
    - Define the internal various types of variables and operations by inheritance
  - Maximal code reusability
    - See the success of PyTorch, TensorFlow, DyNet, …
- You define a hypothesis set by designing a directed acyclic graph.
- The hypothesis space is then a set of all possible parameter settings.

* Functional programming as well ☺

Supervised Learning

- Three points to consider both in research and in practice
  1. How do we decide/design a **hypothesis set**?
  2. How do we decide a **loss function**?
  3. How do we **optimize** the loss function?

A Neural network computes a conditional distribution

- Supervised learning: what is y given x ?

$$f_\theta(x) = ?$$

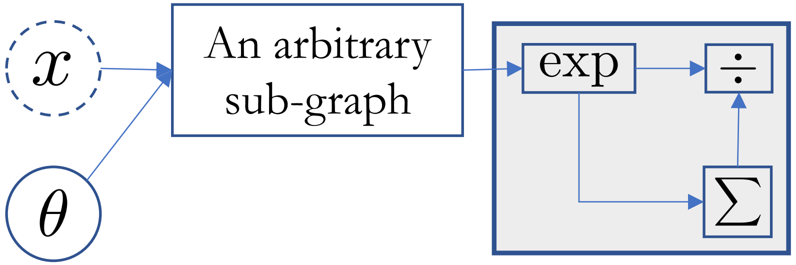
- In other words, how probable is a certain value y' of y given x?

$$p(y = y'|x) = ?$$

- What kind of distributions?
  - Binary classification: Bernoulli distribution
  - Multiclass classification: Categorical distribution
  - Linear regression: Gaussian distribution
  - Multimodal linear regression: Mixture of Gaussians

Important distributions – Categorical

- How probable is a certain value y' of y given x? $p(y = y'|x) = ?$
- Multi-class classification: Categorical distribution $\mathcal{C}(\{\mu_1, \mu_2, \ldots, \mu_C\})$
  - Probability: $p(y = v|x) = \mu_v$, where $\sum_{v=1}^{C} \mu_v = 1$
  - Fully characterized by $\{\mu_1, \mu_2, \ldots, \mu_C\}$.
  - A neural network then should turn the input x into a vector $\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_C \end{bmatrix}$



using a **softmax** function: $\text{softmax}(a) = \frac{1}{\sum_{v=1}^{C} \exp(a_v)} \exp(a)$.

Loss Function – negative log-probability

- Once a neural network outputs a conditional distribution $p_\theta(y|x)$, a natural way to define a loss function arises.
- Make sure training data is maximally likely:
  - Equiv. to making sure each and every training example is maximally likely.

$$\arg\max_\theta \log p_\theta(D) = \arg\max_\theta \sum_{n=1}^{N} \log p_\theta(y_n|x_n)$$

  - Why *log*? – many reasons… but out of the lecture's scope.
- Equivalently, we want to minimize the *negative* log-probability.
  - A loss function is the sum of negative log-probabilities of correct answers.

$$L(\theta) = \sum_{n=1}^{N} l(M_\theta(x_n), y_n) = -\sum_{n=1}^{N} \log p_\theta(y_n|x_n)$$

Supervised Learning

- Three points to consider both in research and in practice
    1. How do we decide/design a **hypothesis set**?
    2. How do we decide a **loss function**?
    3. How do we **optimize** the loss function?

Loss Minimization

- What we now know
  1. How to build a neural network with an arbitrary architecture.
  2. How to define a per-example loss as a negative log-probability.
  3. Define a single directed acyclic graph containing both.
- What we now need to know
  1. Choose an optimization algorithm.
  2. How to use the optimization algorithm to estimate parameters θ.

Gradient-based optimization

- A **continuous**, **differentiable*** function $L : \mathbb{R}^d \to \mathbb{R}$
- Given the current value θ_0, how should I move to minimize L?
- Gradient descent
  - The negative gradient of the function: $-\nabla L(\theta_0)$
  - This is only valid in a local neighbourhood of θ_0: take a very small step!

$$\theta = \theta_0 - \eta \nabla L(\theta_0)$$

- Efficient and effective even in the high dimensional space.
  - Can be improved with the second-order information (Hessian and/or FIM)

* Almost everywhere, but not necessarily everywhere

Backward Computation – Backpropagation

- How do we compute the gradient of the loss function?

1. Manual derivation
    - Relatively doable when the DAG is small and simple.
    - When the DAG is larger and complicated, too much hassle.
2. Automatic differentiation (autograd)
    - Use the chain rule of derivatives

$$\frac{\partial (f \circ g)}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x}$$

    - The DAG is nothing but a composition of (mostly) differentiable functions.
    - Automatically apply the chain rule of derivatives.

Backward Computation – Backpropagation

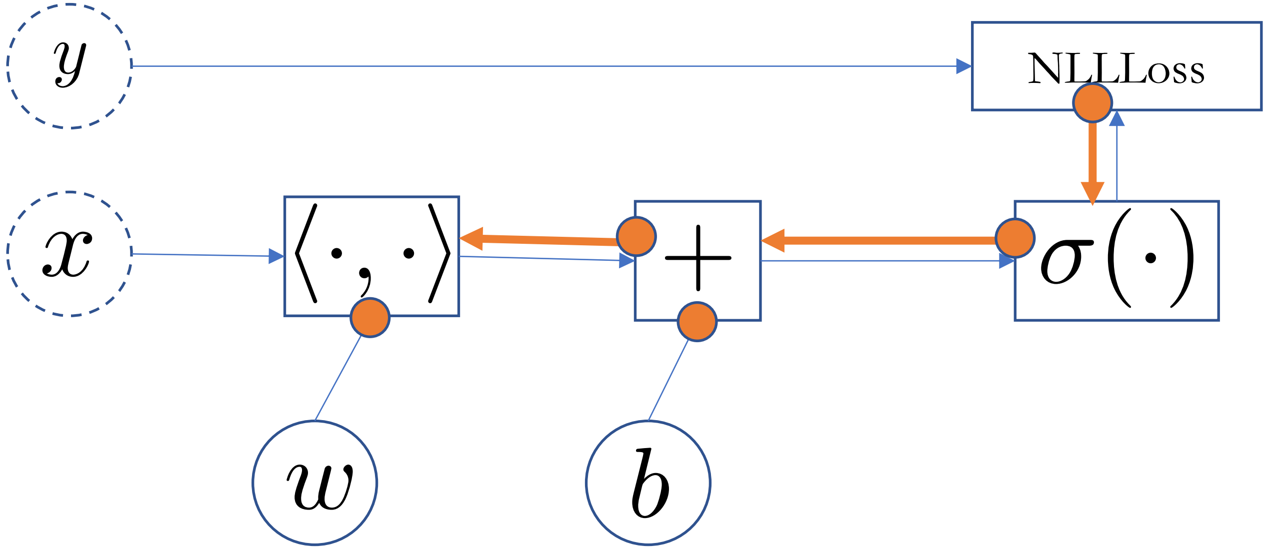
- Automatic differentiation (autograd)
  1. Implement the Jacobian-vector product of each OP node:

$$\begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \vdots \\ \frac{\partial L}{\partial x_d} \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_{d'}}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_1}{\partial x_d} & \cdots & \frac{\partial F_{d'}}{\partial x_d} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial F_1} \\ \vdots \\ \frac{\partial L}{\partial F_{d'}} \end{bmatrix}$$

     - Can be implemented efficiently without explicitly computing the Jacobian.
     - The same implementation can be reused every time the OP node is called.

Backward Computation – Backpropagation

- Automatic differentiation (autograd)
  2. Reverse-sweep the DAG starting from the loss function node.
     - Iteratively multiplies the Jacobian of each OP node until the leaf nodes of the parameters.
     - As expensive as forward computation with a constant overhead: O(N), where N: # of nodes.



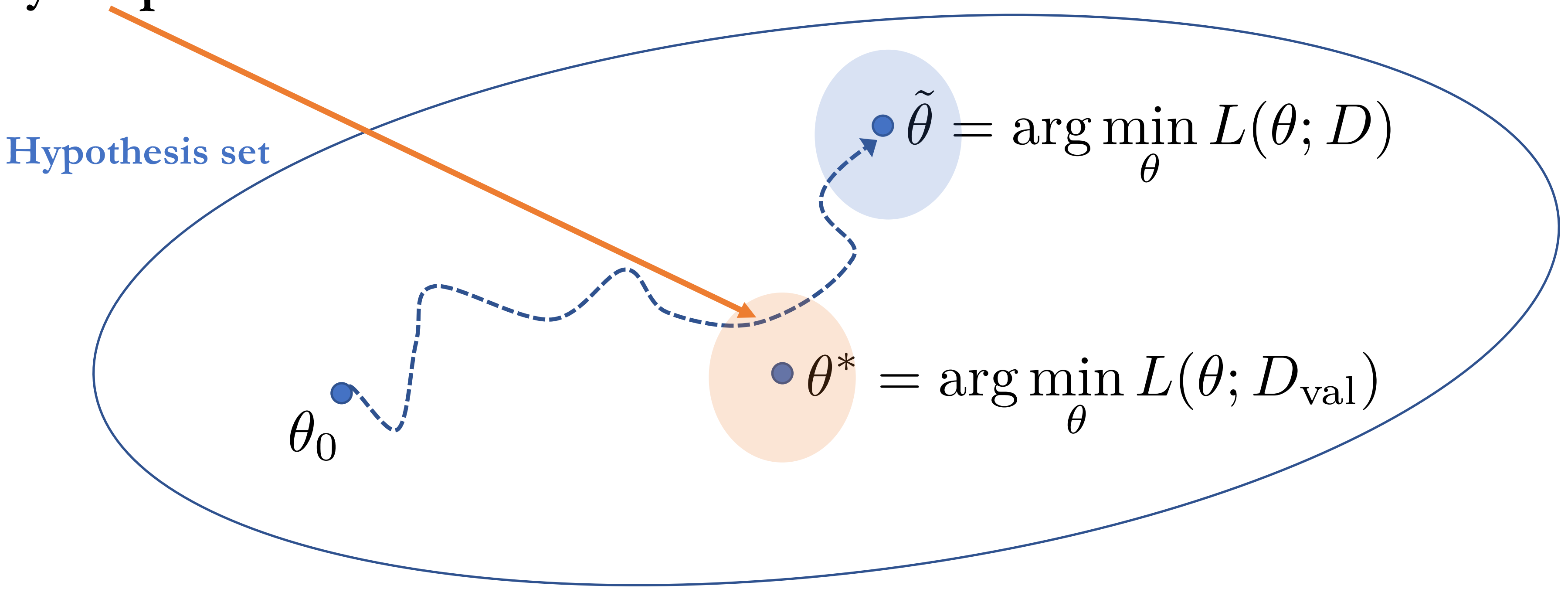
Gradient-based Optimization

- Backpropagation gives us the gradient of the loss function w.r.t. θ
- Readily used by off-the-shelf gradient-based optimizers
  - Gradient descent, L-BFGS, Conjugate gradient, …
  - Though, most are not applicable in a realistic neural network with 10s or 100s of millions of parameters.
- Stochastic gradient descent
  - Approximate the full loss function (the sum of per-examples losses) using only a small random subset of training examples:

$$\nabla L \approx \frac{1}{N'} \sum_{n=1}^{N'} \nabla l(M(x_{n'}), y_{n'})$$

Stochastic Gradient Descent – Early Stopping

- An efficient way to prevent overfitting
  - Overfitting: the training loss is low, but the validation loss is not.
  - The most serious problem in statistical machine learning.
  - **Early-stop** based on the validation loss



Supervised Learning with Neural Networks

1. How do we decide/design a **hypothesis set**?
   - Design a network architecture as a directed acyclic graph
2. How do we decide a **loss function**?
   - Frame the problem as a conditional distribution modelling
   - The per-example loss function is a negative log-probability of a correct answer
3. How do we **optimize** the loss function?
   - Automatic backpropagation: no manual gradient derivation
   - Stochastic gradient descent with early stopping [and adaptive learning rate]

Language modeling as supervised learning

On the boundary between unsupervised and supervised learning

Text Classification

- Input: a natural language sentence/paragraph
- Output: a category to which the input text belongs
  - There are a fixed number C of categories
- Examples
  - Sentiment analysis: is this review positive or negative?
  - Text categorization: which category does this blog post belong to?
  - Intent classification: is this a question about a Chinese restaurant?

How to represent a sentence

- A sentence is a variable-length sequence of tokens: $X = (x_1, x_2, \ldots, x_T)$
- Each token could be any one from a vocabulary: $x_t \in V$
- Examples
  - (케이프, 타운에서, 강의, 중, 입니다, .)
    - Vocabulary: All unique, space-separated tokens in Korean
  - (케이프, 타운, 에서, 강의, 중, 입니다, .)
    - Vocabulary: All uniqued, segmented tokens in Korean
  - (케,이,프, ,타,운, 에, 서, [], 강, 의, [], 중, [], 입, 니, 다, .)
    - Vocabulary: All Korean syllables
  - And many more possibilities…

How to represent a sentence

- A sentence is a variable-length sequence of tokens: $X = (x_1, x_2, \ldots, x_T)$
- Each token could be any one from a vocabulary: $x_t \in V$
- Once the vocabulary is fixed and encoding is done, a sentence or text is just a sequence of "integer indices".
- Examples:
  - (케이프, 타운, 에서, 강의, 중, 입니다, .)
  - (8398, 2301, 20, 288, 12, 19, 5)

$V =$

Index	Token
5	.
12	중
19	입니다
20	에서
…	…
288	강의
827	재단
…	…

How to represent a token

- A token is an integer “index”.
- How do should we represent a token so that it reflects its “meaning”?
- First, we assume nothing is known: use an one-hot encoding.

$$x = [0, 0, 0, \dots, 0, 1, 0, \dots, 0] \in \{0, 1\}^{|V|}$$

  - $|V|$: the size of vocabulary
  - Only one of the elements is 1: $\sum_{i=1}^{|V|} x_i = 1$
- Every token is equally distant away from all the others.

$$\|x - y\| = c > 0, \text{if } x \neq y$$

How to represent a token

- How do should we represent a token so that it reflects its "meaning"?
- First, we assume nothing is known: use a one-hot encoding.
- Second, the neural network capture the token's meaning as a vector.
- This is done by a simple matrix multiplication:
  $Wx = W\left[\hat{x}\right]$, if x is one-hot,
  where $\hat{x} = \arg\max_j x_j$ is the token's index in the vocabulary.

Table Lookup

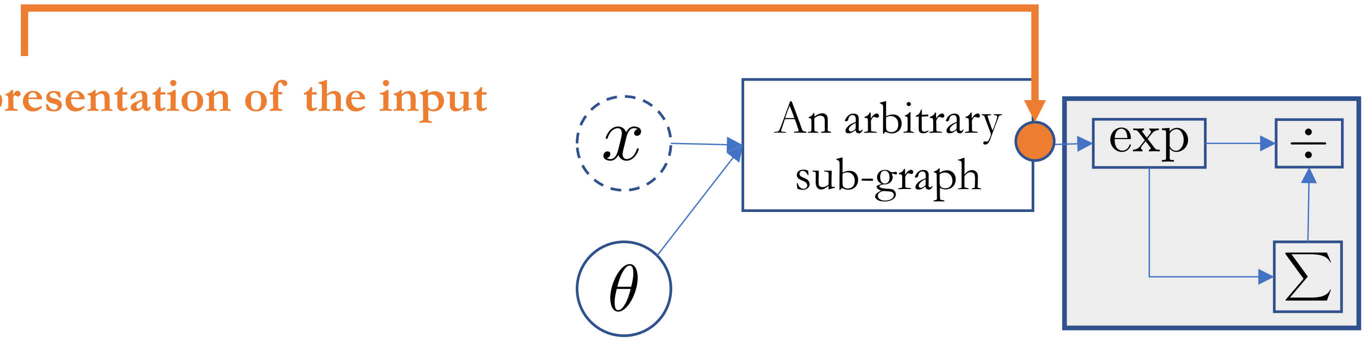
$$W^\top = \begin{bmatrix} w_{11} & \cdots & w_{1d} \\ \vdots & \ddots & \vdots \\ \boxed{w_{\hat{x}1} \quad \cdots \quad w_{\hat{x}d}} \\ \vdots & \ddots & \vdots \\ w_{|V|1} & \cdots & w_{|V|d} \end{bmatrix}$$

How to represent a sentence – CBoW

- After the table-lookup operation,* the input sentence is a sequence of continuous, high-dimensional vectors:

  $X = (e_1, e_2, \ldots, e_T)$, where $e_t \in \mathbb{R}^d$
- The sentence length T differs from one sentence to another.
- The classifier needs to eventually compress it into a single vector.

A fixed-size vector representation of the input



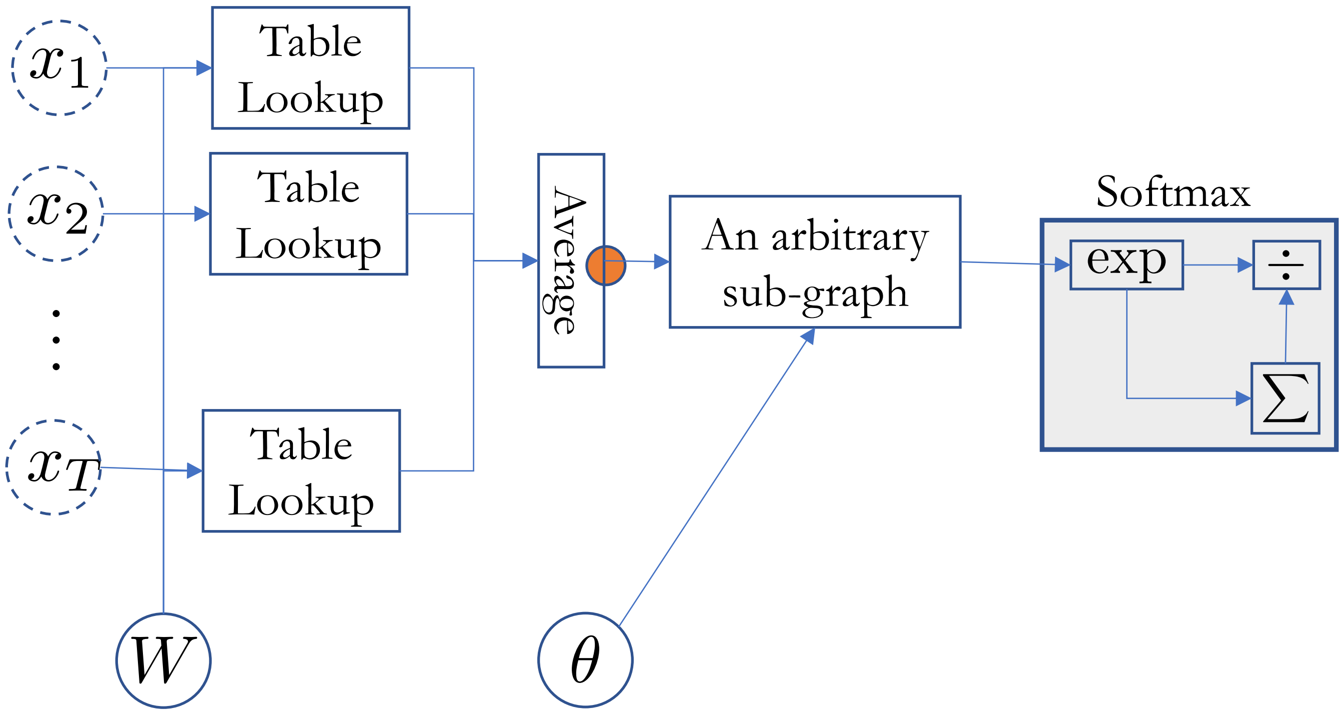
* The table-lookup operation would be one node in the DAG.

How to represent a sentence – CBoW

- Continuous bag-of-words
  - Ignore the order of the tokens: $(x_1, x_2, \ldots, x_T) \rightarrow \{x_1, x_2, \ldots, x_T\}$
  - Simply average the token vectors: $$\frac{1}{T}\sum_{t=1}^{T} e_t$$
    - Averaging is a differentiable operator.
    - Just one operator node in the DAG.
  - Generalizable to bag-of-n-grams
    - N-gram: a phrase of N tokens
- Extremely effective in text classification [Iyyer et al., 2016; Cho, 2017; and many more]
  - For instance, if there are many positive words, the review is likely positive.
- In practice, use FastText [Bojanowski et al., 2017]

How to represent a sentence – CBoW

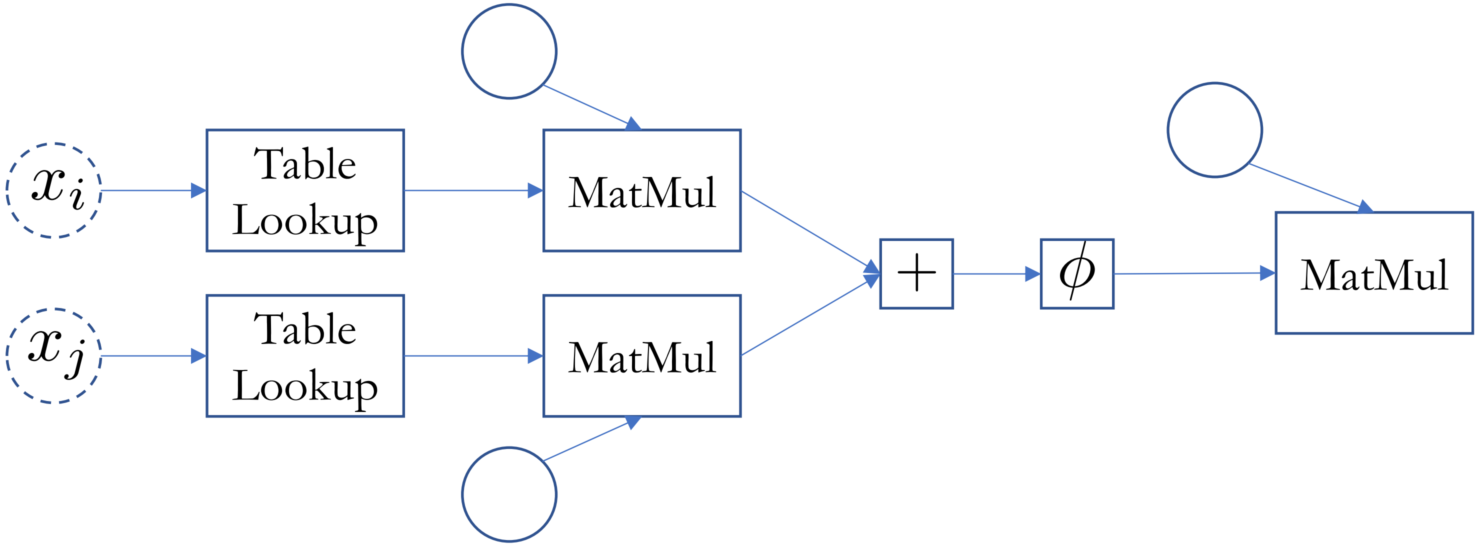
- Continuous bag-of-words based multi-class text classifier



- With this DAG, you use automatic backpropagation and stochastic gradient descent to train the classifier.

How to represent a sentence – RN

- Relation Network [Santoro et al., 2017]: Skip Bigrams
  - Consider all possible pairs of tokens: $(x_i, x_j), \forall i \neq j$
  - Combine two token vectors with a neural network for each pair
    $f(x_i, x_j) = W\phi(U_{\text{left}} e_i + U_{\text{right}} e_j)$
    - ϕ is a element-wise nonlinear function, such as tanh or ReLU ($\max(0, a)$)
    - One subgraph in the DAG.

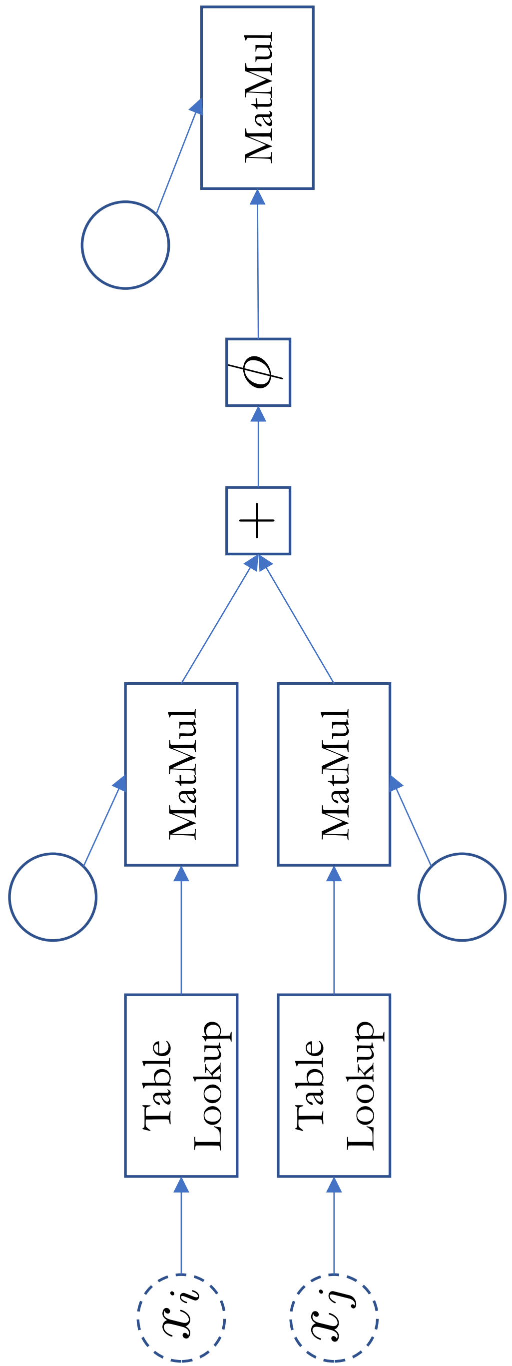


How to represent a sentence – RN

- Relation Network: Skip Bigrams
  - Considers all possible pairs of tokens: $(x_i, x_j), \forall i \neq j$
    $f(x_i, x_j) = W\phi(U_{\text{left}} e_i + U_{\text{right}} e_j)$
  - Considers the "relation"ship between each pair of words
  - Averages all these relationship vectors

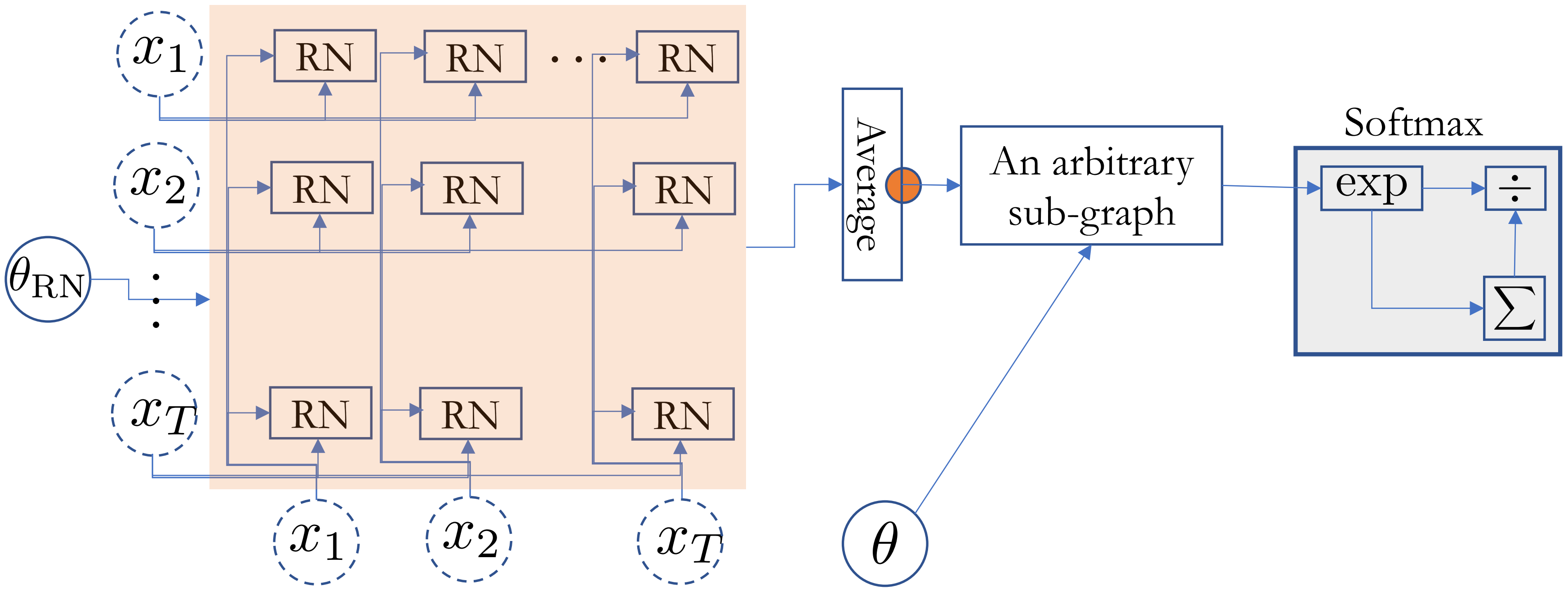
$$\text{RN}(X) = \frac{1}{2N(N-1)} \sum_{i=1}^{T-1} \sum_{j=i+1}^{T} f(x_i, x_j)$$

  - Could be generalized to triplets and so on at the expense of computational efficient.



How to represent a sentence – RN

- Relation Network: Skip Bigrams
  - Considers all possible pairs of tokens: $(x_i, x_j), \forall i \neq j$
    $f(x_i, x_j) = W\phi(U_{\text{left}} e_i + U_{\text{right}} e_j)$
  - Considers the pair-wise "relation"ship $\text{RN}(X) = \frac{1}{2N(N-1)} \sum_{i=1}^{T-1} \sum_{j=i+1}^{T} f(x_i, x_j)$
  - Averages all these relationship vectors

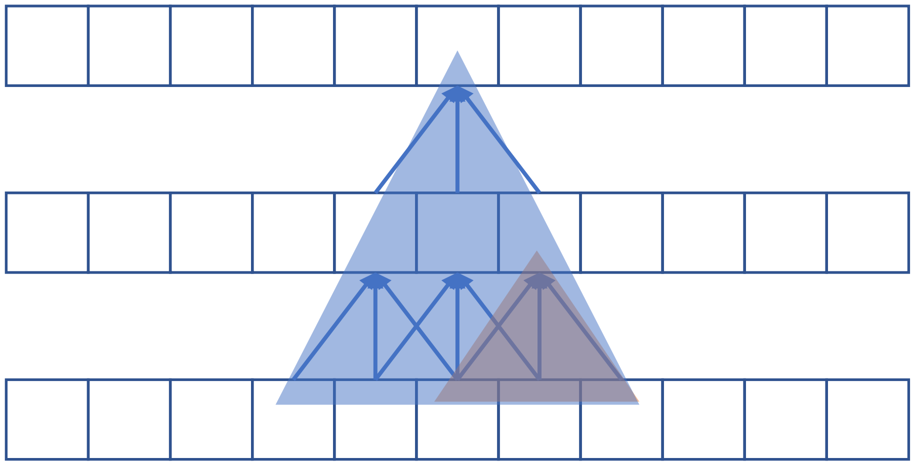


How to represent a sentence – CNN

- Convolutional Networks [Kim, 2014; Kalchbrenner et al., 2015]
  - Captures k-grams hierarchically
  - One 1-D convolutional layer: considers all k-grams

$$h_t = \phi\left(\sum_{\tau=-k/2}^{k/2} W_\tau e_{t+\tau}\right), \text{ resulting in } H = (h_1, h_2, \ldots, h_T).$$

  - Stack more than one convolutional layers: progressively-growing window
  - Fits our intuition of how sentence is understood: **tokens→multi-word expressions→phrases→sentence**



How to represent a sentence – CNN

- Convolutional Networks [Kim, 2014; Kalchbrenner et al., 2015]
  - Captures k-grams hierarchically
  - Stack more than one convolutional layers: progressively-growing window
  - **tokens→multi-word expressions→phrases→sentence**
- In practice, just another operation node in a DAG:
  - Extremely efficient implementations are available in all of the major frameworks.
- Some considerations
  - Multi-width convolutional layers [Kim, 2014; Lee et al., 2017]
  - Dilated convolutional layers [Kalchbrenner et al., 2016]
  - Gated convolutional layers [Gehring et al., 2017]

How to represent a sentence – Self-Attention

- Can we combine and generalize the relation network and the CNN?
- Relation Network:
  - Each token's representation is computed against all the other tokens

$$h_t = f(x_t, x_1) + \cdots + f(x_t, x_{t-1}) + f(x_t, x_{t+1}) + \cdots + f(x_t, x_T)$$

- CNN:
  - Each token's representation is computed against neighbouring tokens

$$h_t = f(x_t, x_{t-k}) + \cdots + f(x_t, x_t) + \cdots + f(x_t, x_{t+k})$$

- RN considers the entire sentence vs. CNN focuses on the local context.

How to represent a sentence – Self-Attention

- Can we combine and generalize the relation network and the CNN?
- CNN as a weighted relation network:
  - Original: $h_t = f(x_t, x_{t-k}) + \cdots + f(x_t, x_t) + \cdots + f(x_t, x_{t+k})$
  - Weighted:

$$h_t = \sum_{t'=1}^{T} \mathbb{I}(|t' - t| \leq k) f(x_t, x_{t'})$$

where $\mathbb{I}(S) = 1, \text{ if } S \text{ is true, and } 0, \text{otherwise}$.

- Can we compute those weights instead of fixing them to 0 or 1?

How to represent a sentence – Self-Attention

- Can we compute those weights instead of fixing them to 0 or 1?
- That is, compute the weight of each pair $(x_t, x_{t'})$

$$h_t = \sum_{t'=1}^{T} \alpha(x_t, x_{t'}) f(x_t, x_{t'})$$

- The weighting function could be yet another neural network
  - Just another subgraph in a DAG: easy to use!

    $$\alpha(x_t, x_{t'}) = \sigma(\mathrm{RN}(x_t, x_{t'})) \in [0, 1]$$

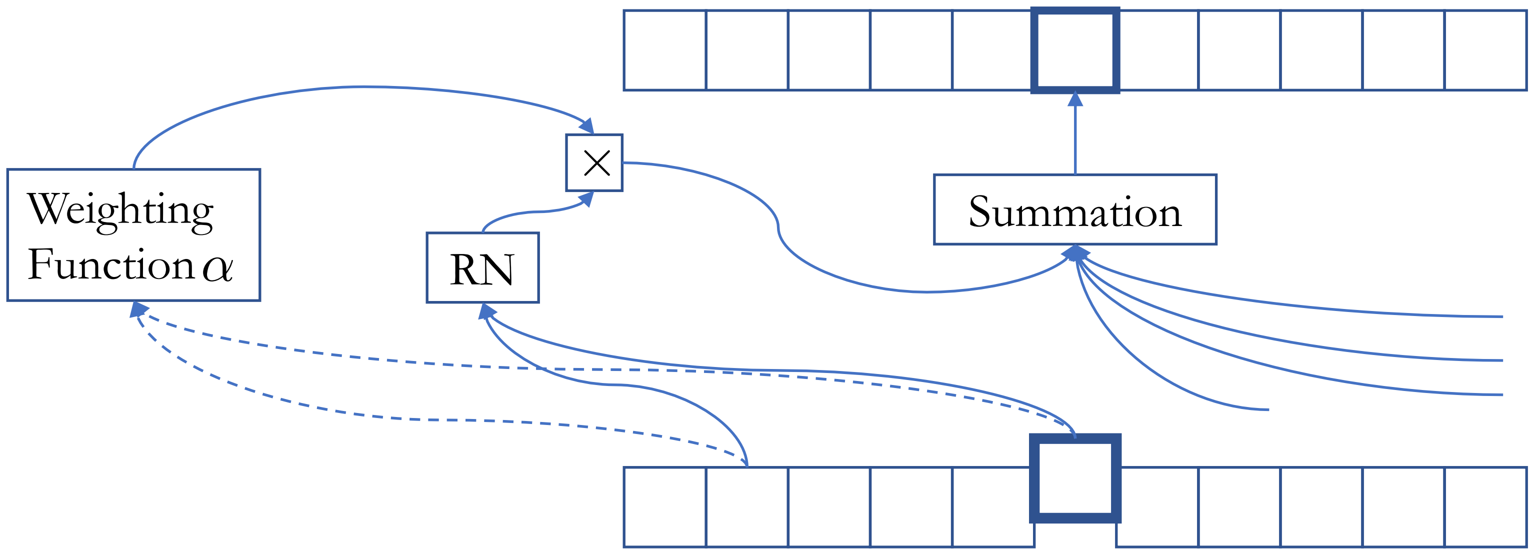
  - Perhaps we want to normalize them so that the weights sum to one

    $$\alpha(x_t, x_{t'}) = \frac{\exp(\beta(x_t, x_{t'}))}{\sum_{t''=1}^{T} \exp(\beta(x_t, x_{t''}))}, \text{where } \beta(x_t, x_{t'}) = \mathrm{RN}(x_t, x_{t'}))$$

[Bahdanau et al., 2015; Vaswani et al., 2017; Parikh et al., 2016]

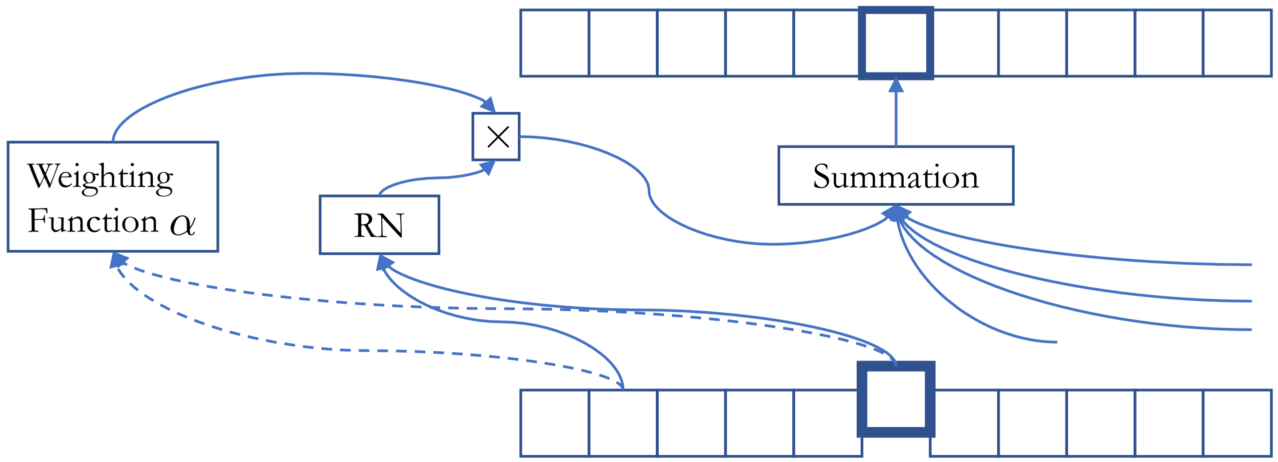
How to represent a sentence – Self-Attention

- Self-Attention: a generalization of CNN and RN.
- Able to capture long-range dependencies within a single layer.
- Able to ignore irrelevant long-range dependencies.



How to represent a sentence – Self-Attention

- Self-Attention: a generalization of CNN and RN.
- Able to capture long-range dependencies within a single layer.
- Able to ignore irrelevant long-range dependencies.
- Further generalization via multi-head and multi-hop attention



How to represent a sentence – RNN

- Weaknesses of self-attention
  1. Quadratic computational complexity $O(T^2)$
  2. Some operations cannot be done easily: e.g., counting, …
- Online compression of a sequence $O(T)$

  $h_t = \mathrm{RNN}(h_{t-1}, x_t)$, where $h_0 = 0$.
- Memory h_t allows it to be Turing complete.*

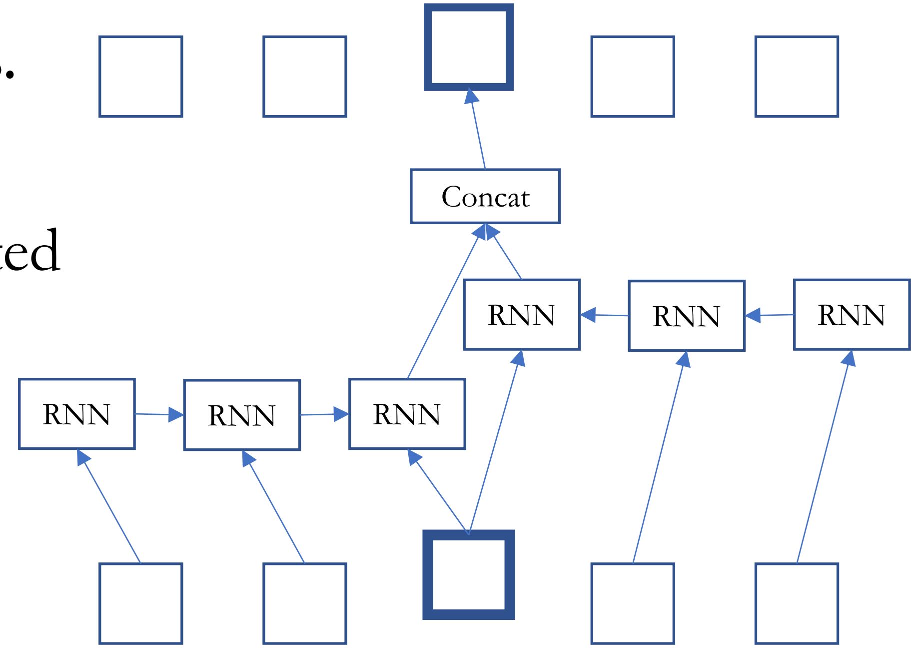
* Under some extreme assumptions…

How to represent a sentence – RNN

- Recurrent neural network: online compression of a sequence $O(T)$

  $h_t = \mathrm{RNN}(h_{t-1}, x_t)$, where $h_0 = 0$.

- Bidirectional RNN to account for both sides.
- Inherently sequential processing
  - Less desirable for modern, parallelized, distributed computing infrastructure.
- LSTM [Hochreiter&Schmidhuber, 1999] and GRU [Cho et al., 2014] have become de facto standard
  - All standard frameworks implement them.
  - Efficient GPU kernels are available.



How to represent a sentence

- We have learned five ways to extract a sentence representation:
  - In all but CBoW, we end up with a set of vector representations. $H = \{h_1, \dots, h_T\}$
  - These approaches could be "stacked" in an arbitrary way to improve performance.
    - Chen, Firat, Bapna et al. [2018] combine self-attention and RNN to build the state-of-the-art machine translation system.
    - Lee et al. [2017] stack RNN on top of CNN to build an efficient fully character-level neural translation system.
    - Because all of these are differentiable, the same mechanism (backprop+SGD) works as it is for any other machine learning model.
  - These vectors are often averaged/max-pooled for classification.

So far, we have learned…

- Token representation
  - How do we represent a discrete token in a neural network?
  - Training this neural network leads to so-called **continuous word embedding**.
- Sentence representation
  - How do we extract useful representation from a sentence?
  - We learned five different ways to do so: CBoW, RN, CNN, Self-Attention, RNN

- Questions?

Language Modelling

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

Autoregressive language modelling

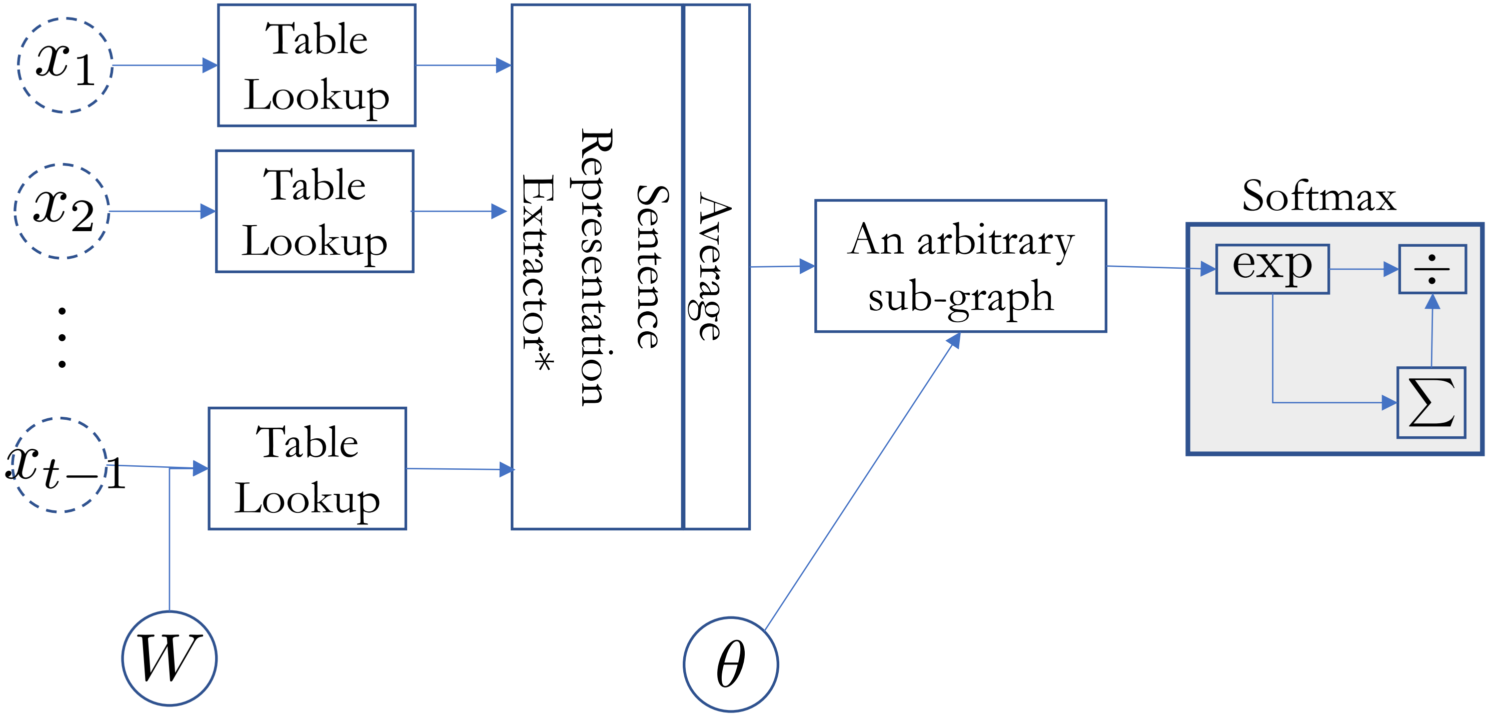
- Autoregressive sequence modelling
  - The distribution over the next token is based on all the previous tokens.
    $p(X) = p(x_1)p(x_2|x_1)\cdots p(x_T|x_1,\ldots,x_{T-1})$
  - This equality holds exactly due to the def. of conditional distribution.
- Unsupervised learning becomes a set of supervised 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.

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:

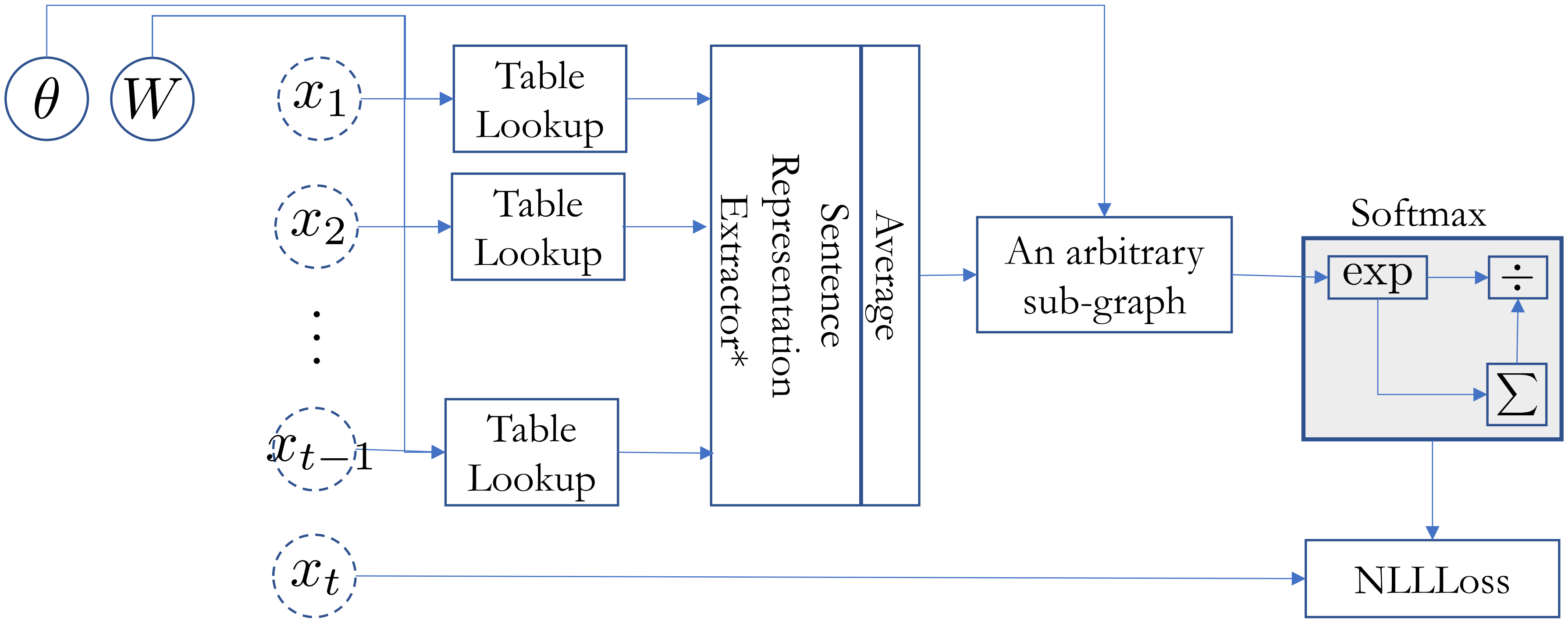


* See Lecture 2.

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?
- Assume a Markovian property

$$p(X) = \prod_{t=1}^{T} p(x_t | x_{<t}) \approx \boxed{\prod_{t=1}^{T} p(x_t | x_{t-n}, \ldots, x_{t-1})}$$

- This turned out to be crucial, and we will discuss why shortly.

N-Gram Language Models

$$p(X) = \prod_{t=1}^{T} p(x_t|x_{<t}) \approx \prod_{t=1}^{T} p(x_t|x_{t-n},\ldots,x_{t-1})$$

- 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:

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

- 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

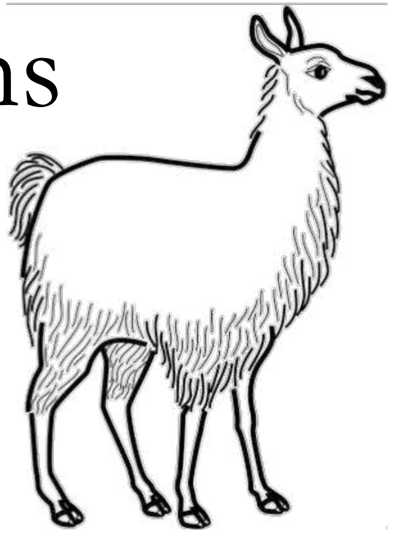
- We need to estimate n-gram probabilities:

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

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

- 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.

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

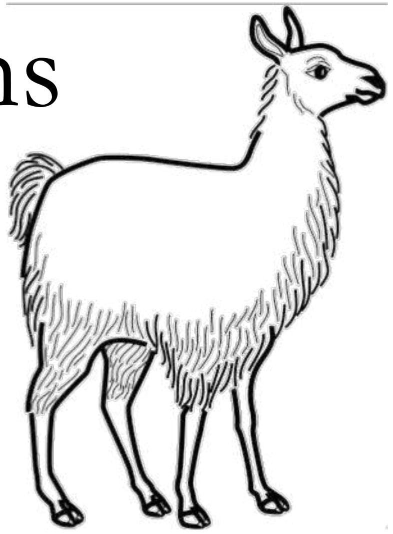
   - The most widely used approach: Kneser-Ney smoothing/backoff
   - **KenLM** implements the efficient n-gram LM model.

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



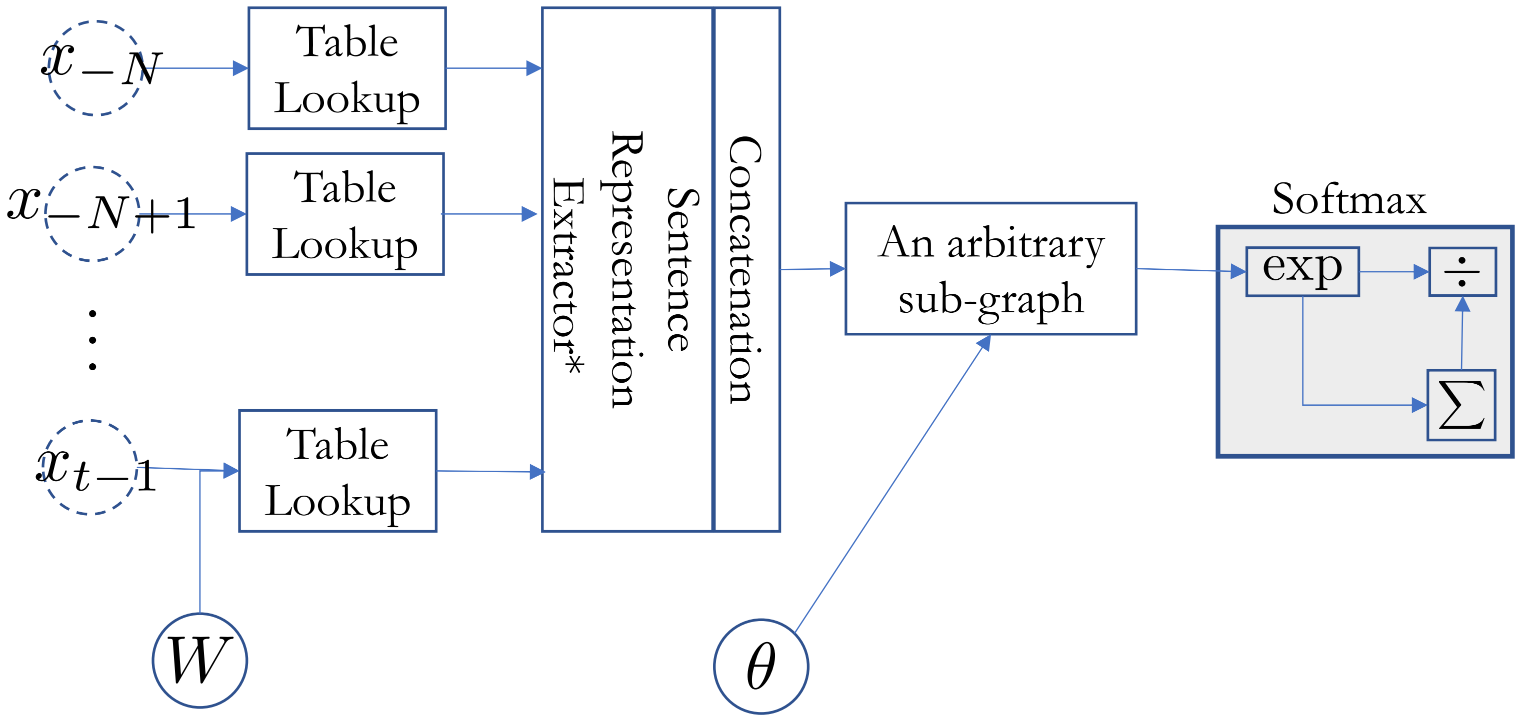
1. Data sparsity: lack of generalization
   - What happens "one" n-gram never happens?

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

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]

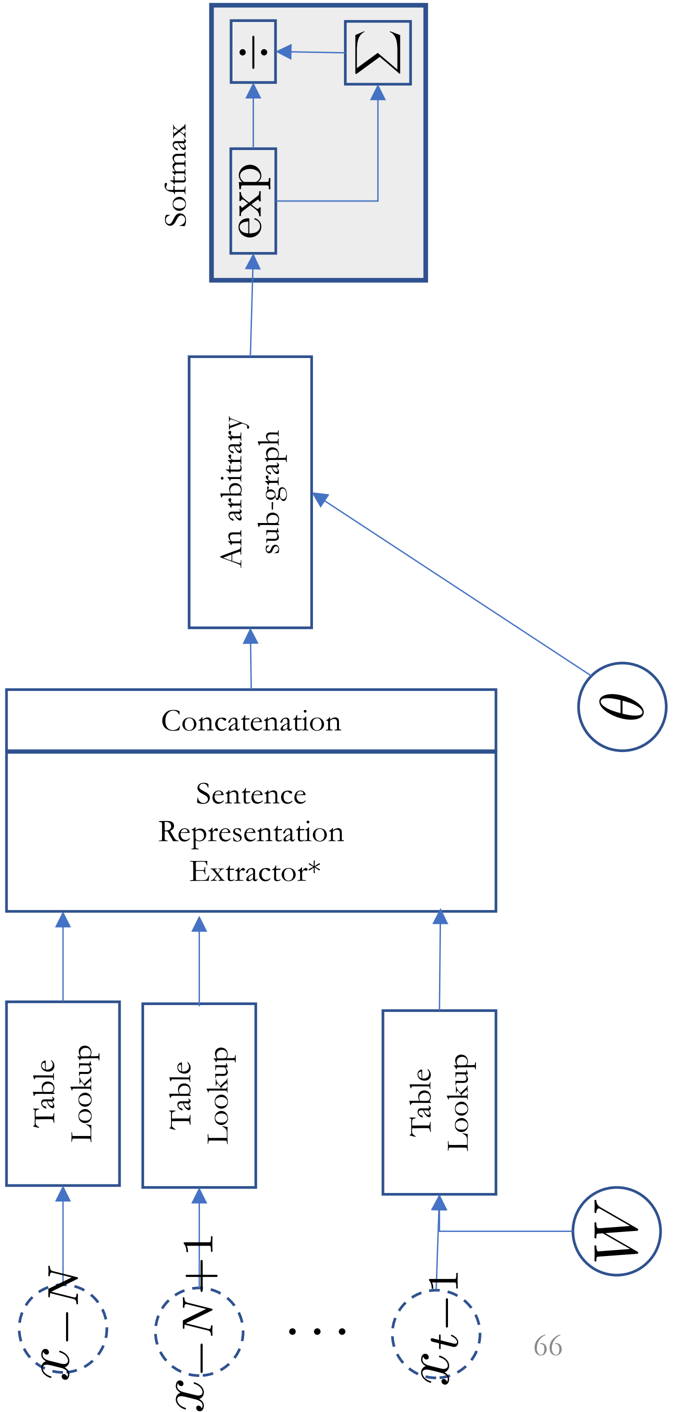
- The first extension of n-gram language models using a neural network



Neural N-Gram Language Model

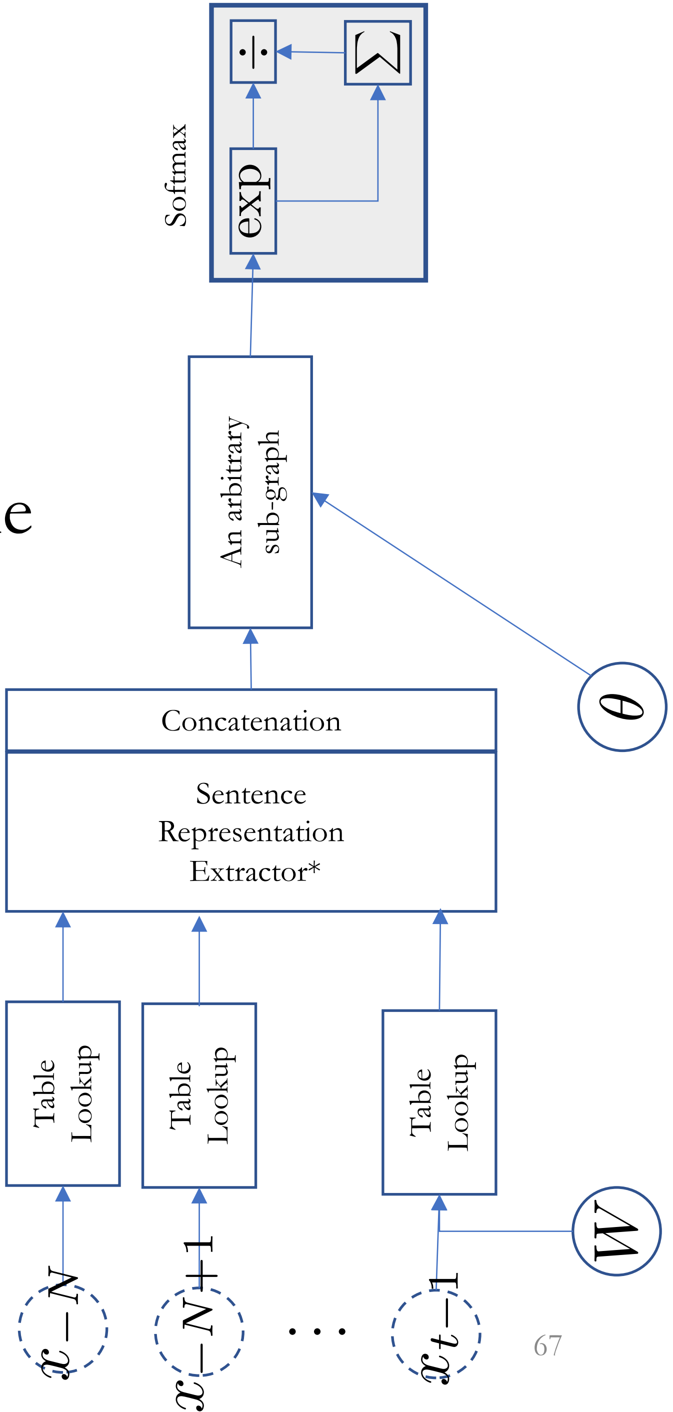
- The first neural language models
- Trained using backpropagation and SGD
- Generalizes to an unseen *n*-gram
- **Addresses the issue of data sparsity**

- *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.

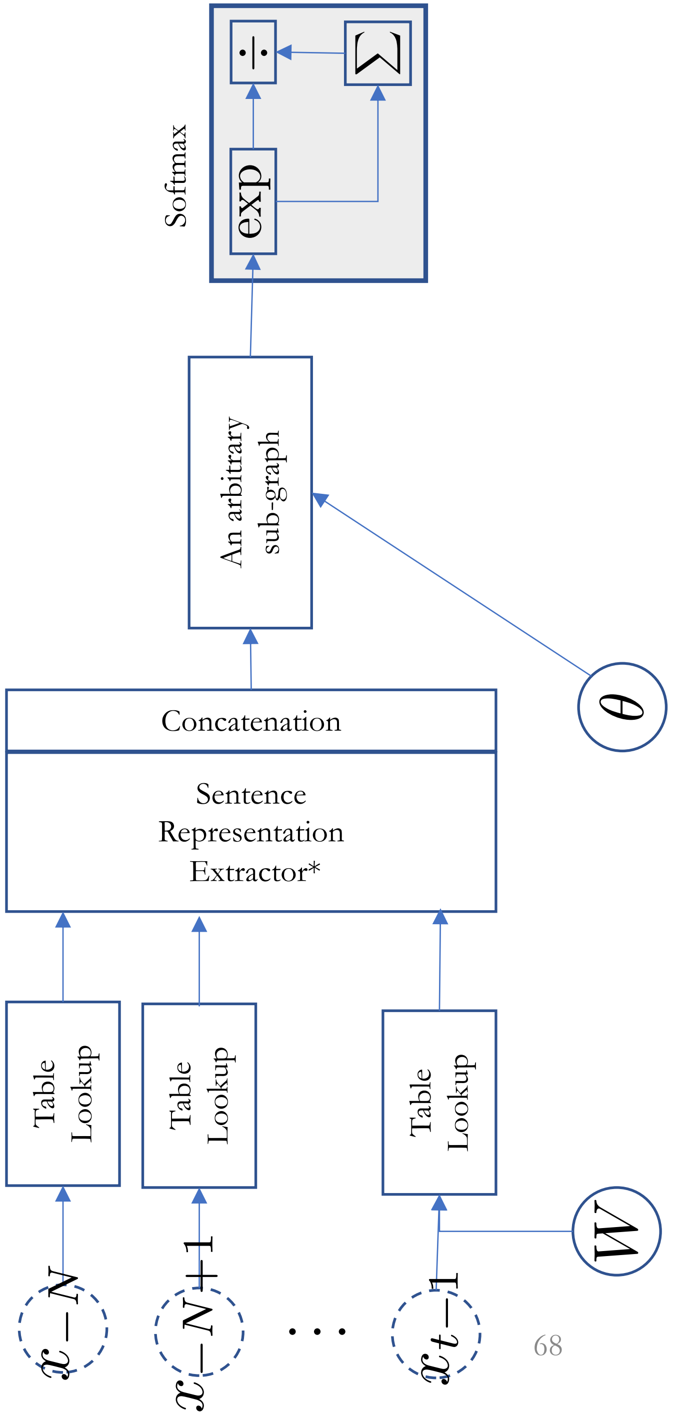


Neural N-Gram Language Model

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

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

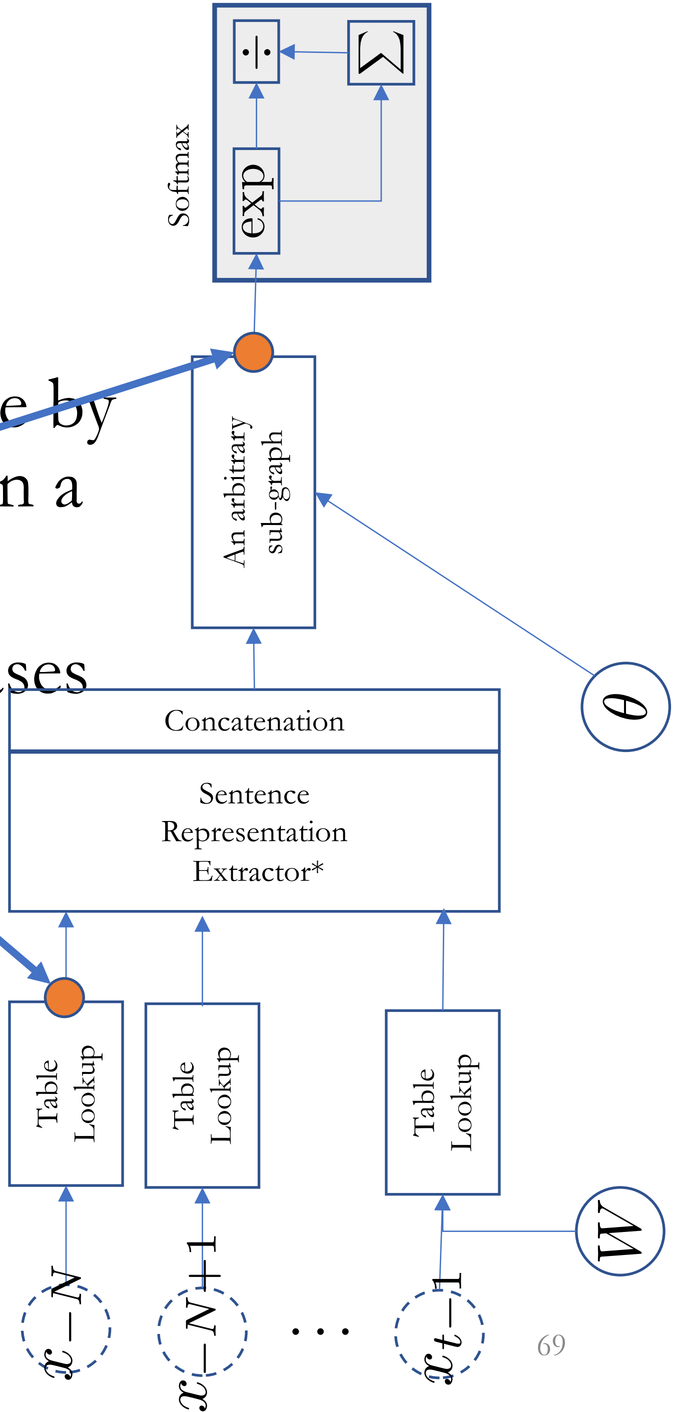
  - If the LM knew “llama” is a mammal similar to “cat”, “dog” and “deer”, it would be able to guess ”chasing a llama” is as likely as “chasing a cat”, “chasing a dog”, and “chasing a deer”.



Neural N-Gram Language Model

- The neural n-gram language model addresses this issue by "learning the similarities" among <u>tokens</u> and <u>phrases</u> 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:

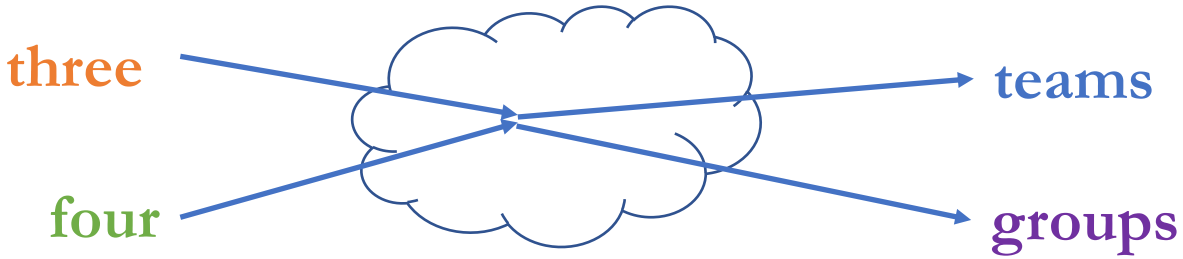
$$D(x_t|x_{t-N},\ldots,x_{t-1}\|x_t|x'_{t-N},\ldots,x'_{t-1}) < \epsilon$$

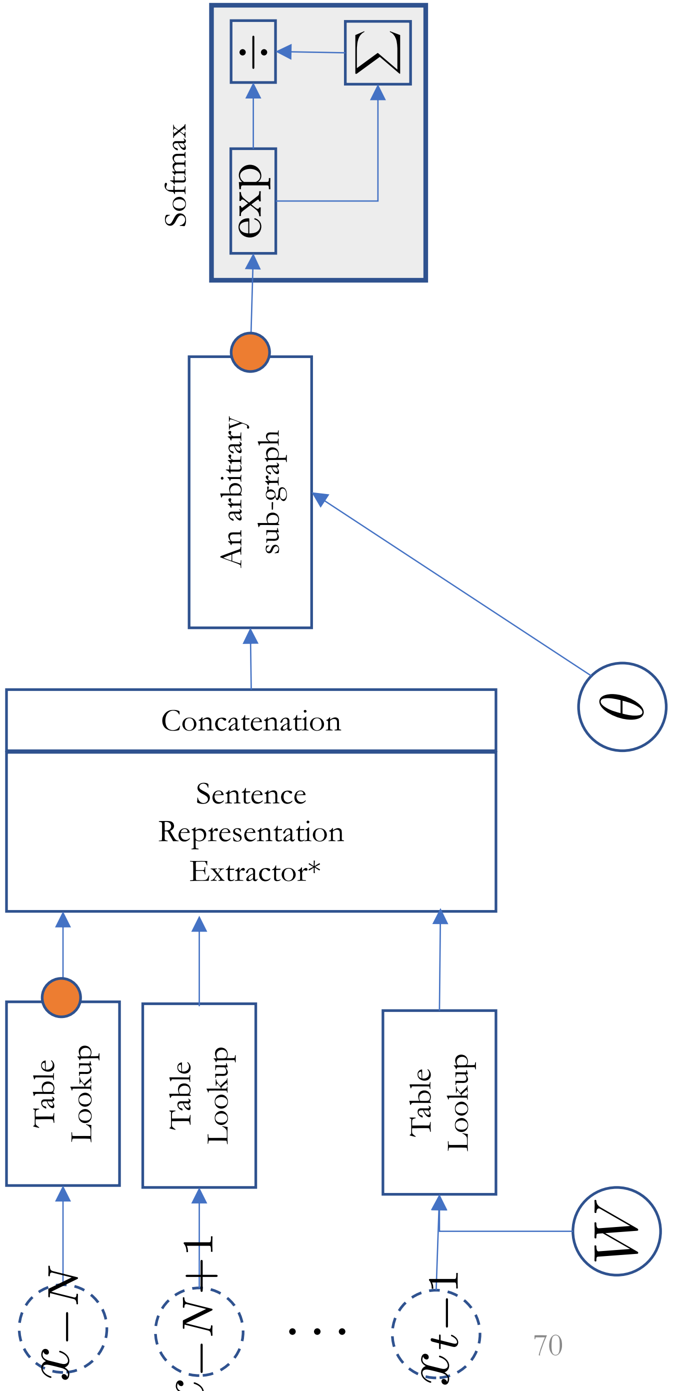


Neural N-Gram Language Model

- 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



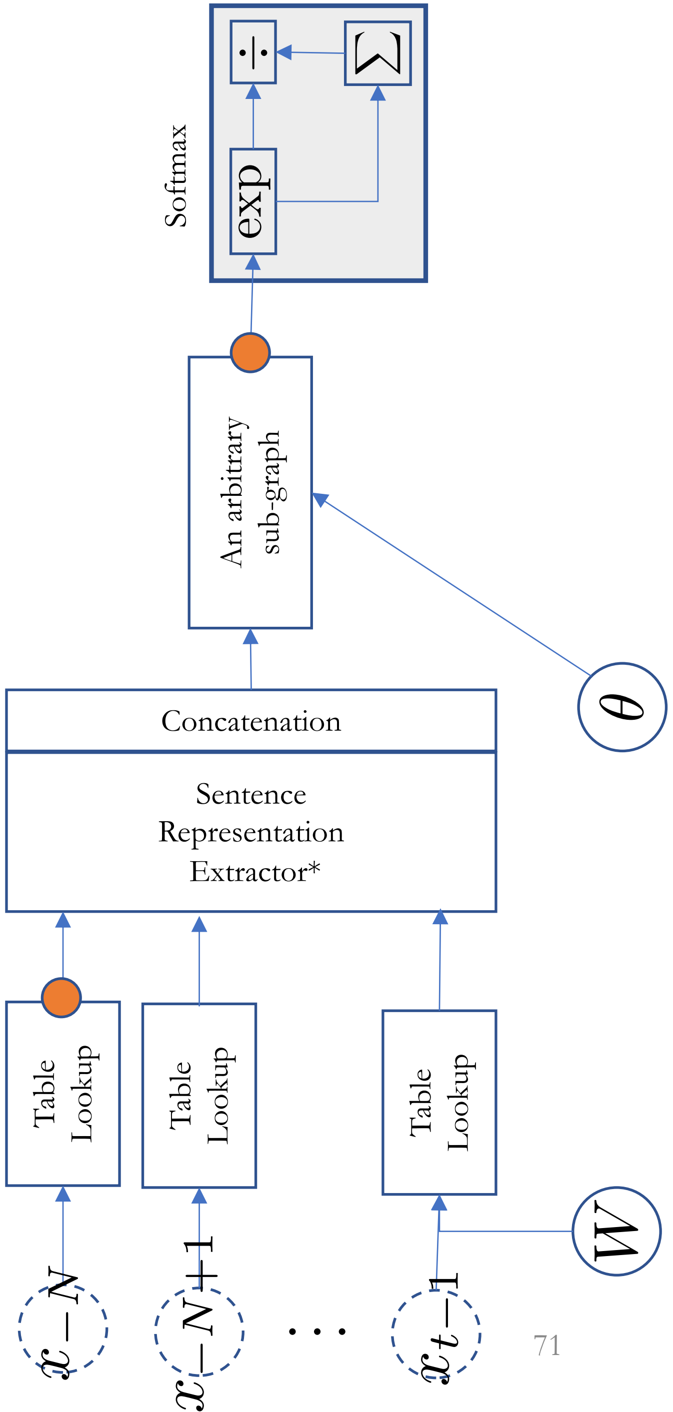


Neural N-Gram Language Model

- 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.

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

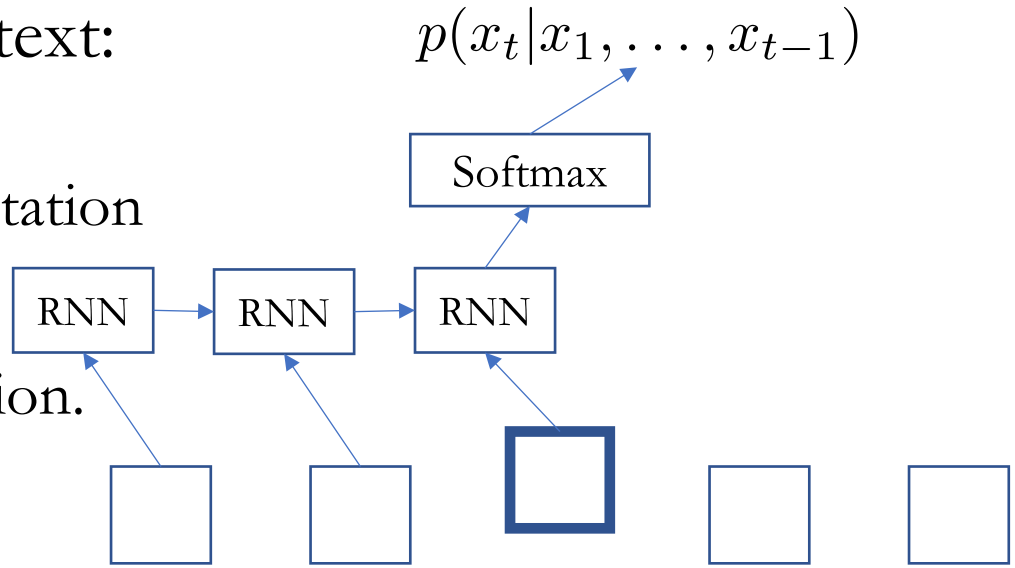


Infinite context $n \longrightarrow \infty$ – CBoW Language Models

- 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.

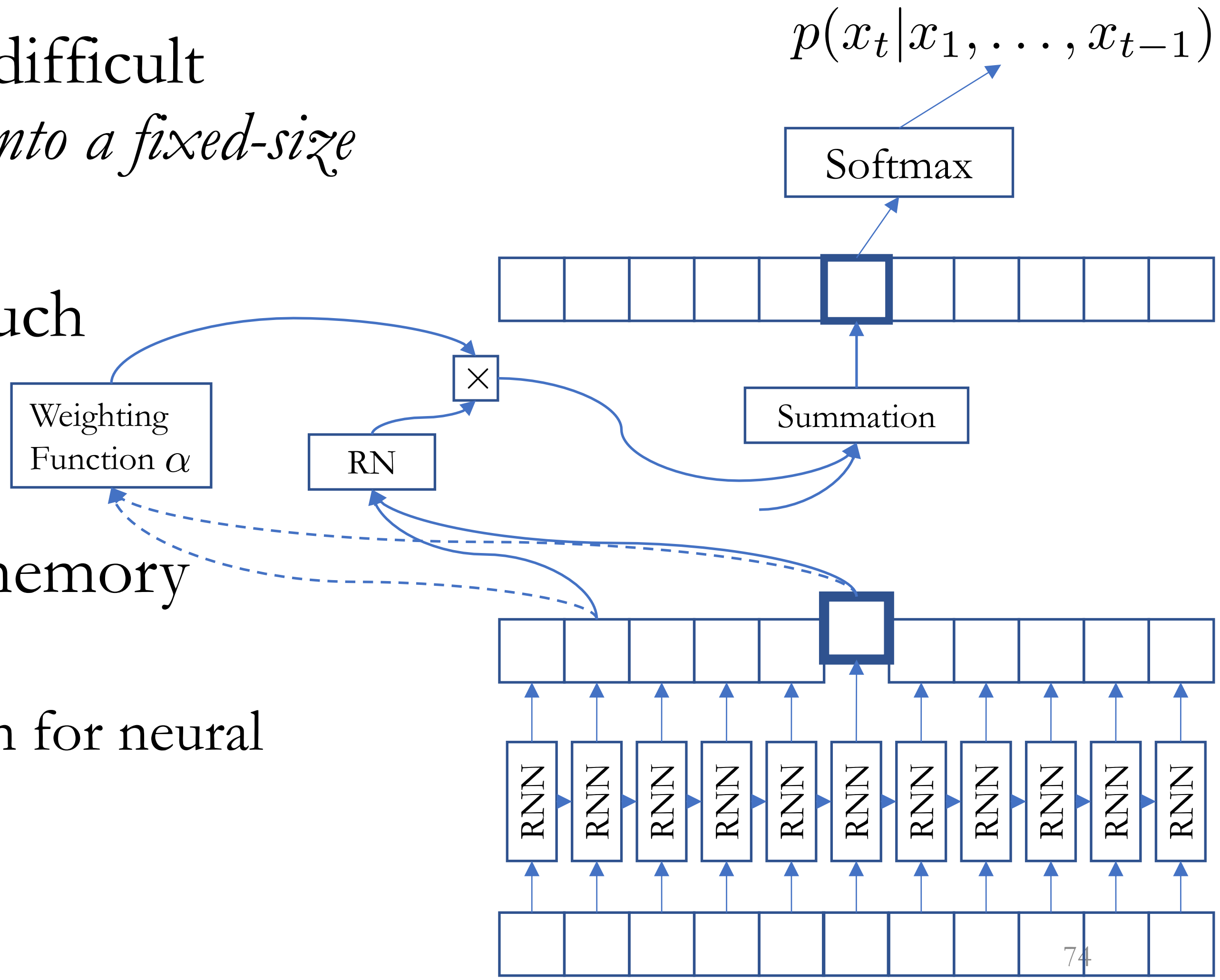
Infinite context n⟶∞ – Recurrent Language Models [Mikolov et al., 2010]

- 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..