Multilingual NLP

2. Neural Language Modeling & Tokenization

Prof. Dr. Goran Glavaš
Center for AI and Data Science (CAIDAS), Uni Würzburg

Image: Alexander Mikhalchyk

After this lecture, you'll...

- Get the core idea behind neural language modeling
- Learn about different ways to tokenize text
- Understand the details of most popular tokenization strategies used in (Transformer-based) neural language models

Content

- **What is Language Modeling?**
- Neural Language Modeling

- Tokenization
  - Word-Level vs. Character-Level
  - Subwords

What is Language Modeling?

- Guess the next word in the sequence:

  I see three people standing on the ___

- Which is more likely?
  - *beach* or *wardrobe*
  - *mountain* or *cat*
  - *sidewalk* or *bookshelf*

What is Language Modeling?

- **Language modeling** aims to determine the probabilities of sequences of words in a language

  P(I see three people standing on the bridge) = ?
  or generally **P**($w_1 w_2 \ldots w_m$) = ?

- Q: How could we estimate this **P**?
- **Maximum likelihood estimation (MLE)**
  - Large corpus **C** = all the text we can get our hands on
  - Count the number of occurrences of c($w_1 w_2 \ldots w_m$) in **C**
  - Divide the count with the size of C

$$\mathbf{P}(w_1 w_2 \ldots w_m) = c(w_1 w_2 \ldots w_m)/|\mathbf{C}|$$

What is Language Modeling?

- **C** = all text ever produced in the history of human kind
  P(*an unicorn riding a crocodile while drinking wine*) = ?

Write a five-sentence story about an unicorn riding a crocodile while drinking wine

Once upon a time, in a land where mythical creatures roamed free, a curious unicorn named Stardust decided to try something daring. While strolling along the banks of the mystical Crocoglide River, she came across a friendly crocodile named Chomper, sunbathing lazily. Stardust, holding a bottle of enchanted wine, struck a deal with Chomper: in exchange for a ride down the river, she would share her wine. As they glided gracefully through the shimmering water, Stardust sipped her wine, marveling at the harmonious partnership between two unlikely creatures. The extraordinary sight of Stardust riding Chomper soon became a legend, a testament to the power of friendship and the boundless possibilities of a world where anything can happen.

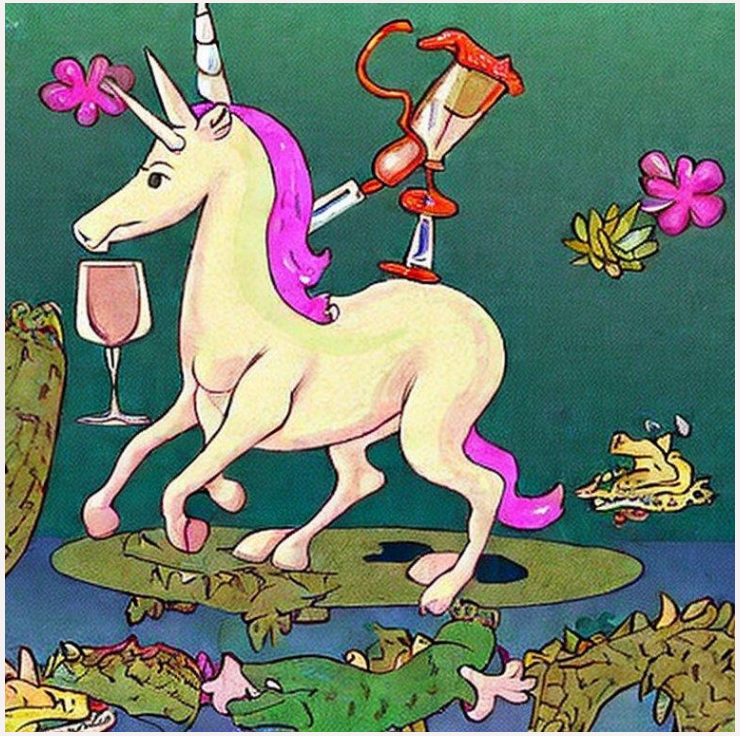


Image generated with Stable Diffusion:
https://stablediffusionweb.com/#demo

N-gram Language Modeling

- Symbolic NLP: every word/token a different symbol
  - Meaning of text stems from the set of symbols it contains

- Under such a representation, language is very sparse
  - The longer the sequence $w_1 w_2 \ldots w_n$ (i.e., the larger n is), the less likely it is that it will be observed in **C**, no matter the size of **C**

- **N-gram language models** are (a partial) remedy

$$P(w_1 w_2 \ldots w_m) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1 w_2) \cdot \ldots \cdot P(w_m|w_1 w_2 \ldots w_{m-1})$$

$$P(w_m|w_1 w_2 \ldots w_{m-1}) \approx P(w_m|w_{m-n+1} \ldots w_{m-1})$$

N-gram Language Modeling

- **N-gram language models** are (a partial) remedy

$$P(w_1 w_2 \ldots w_m) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1 w_2) \cdot \ldots \cdot P(w_m|w_1 w_2 \ldots w_{m-1})$$

$$P(w_m|w_1 w_2 \ldots w_{m-1}) \approx P(w_m|w_{m-n+1} \ldots w_{m-1})$$

  - Unigram LM: $n = 1$
    - $P(w_m|w_1 w_2 \ldots w_{m-1}) \approx P(w_m)$
    - $P(w_1 w_2 \ldots w_m) \approx P(w_1) \cdot P(w_2) \cdot \ldots \cdot P(w_{m-1}) \cdot P(w_m)$
  - Bigram LM: $n = 2$
    - $P(w_m|w_1 w_2 \ldots w_{m-1}) \approx P(w_m \mid w_{m-1})$
    - $P(w_1 w_2 \ldots w_m) \approx P(w_1) \cdot P(w_2|w_1) \cdot \ldots \cdot P(w_{m-1}|w_{m-2}) \cdot P(w_m|w_{m-1})$

N-gram Language Modeling

- Shortcomings of n-gram LM-ing (i.e., symbolic LM-ing)
  - Generating language with n-gram LMs with small n leads to incoherent (non-sensical) text
    - E.g., n = 3, we start with „a blue unicorn"
      - argmax_w P(w|"blue unicorn") → e.g., t-shirt
      - argmax_w P(w|"unicorn t-shirt") → e.g., shop
      - argmax_w P(w|"t-shirt shop") → e.g., bankruptcy

      „a blue unicorn t-shirt shop bankruptcy..." **?!**

  - This can be <u>somewhat remedied</u> (but not much) with:
    - Smoothing schemes
    - Searching for „globally" best probability (e.g., w. beam search)

N-gram Language Modeling

- Shortcomings of n-gram LM-ing (i.e., symbolic LM-ing)
  - Sparsity of symbols in language prevents n-gram LMs with large n
  - But large n would only lead to repetition of n-grams from the training set (on which we estimated conditional probabilities)

  - No **semantic relations** whatsoever between symbols
    - If we know cat is similar to dog
    - Then observing "I pet a white cat" should affect P(dog | pet a white)

  - The **core idea** of **neural** language modeling:
    - By establishing semantic relations between symbols, we can alleviate the issue of sparsity of language (in terms of symbols)
    - Q: How to determine such semantic relations? By LM-ing ☺

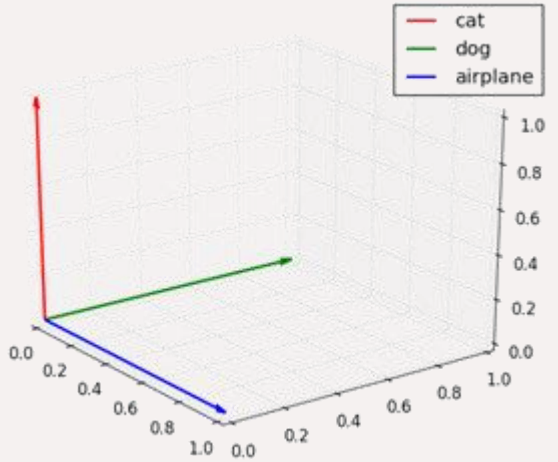
Content

- What is Language Modeling?
- **Neural Language Modeling**

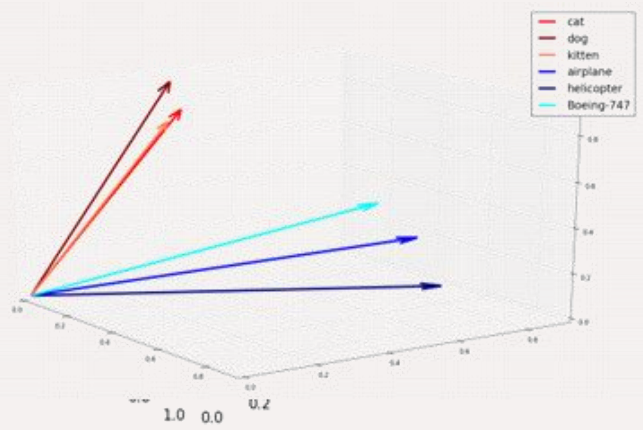
- Tokenization
  - Word-Level vs. Character-Level
  - Subwords

Neural Language Modeling

- Traditional N-gram LM-ing is discrete
  - Any two words equally (dis)similar
  - Vocabulary of N words → corresponds to an N-dim. vector space, each word one axis



- **Neural language modeling** is continuous
  - words → „dense" vectors in continous space
  - Q: how to obtain meaningful vectors?
    - Words with similar meaning get similar vectors
    - Vectors that enable better language modeling



Neural Language Modeling

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*, 1137-1155.

Abstract

A goal of statistical language modeling is to learn the joint probability function of sequences of words in a language. This is intrinsically difficult because of the **curse of dimensionality**: a word sequence on which the model will be tested is likely to be different from all the word sequences seen during training. Traditional but very successful approaches based on n-grams obtain generalization by concatenating very short overlapping sequences seen in the training set. We propose to fight the curse of dimensionality by **learning a distributed representation for words** which allows each training sentence to inform the model about an exponential number of semantically neighboring sentences. The model learns simultaneously (1) a distributed representation for each word along with (2) the probability function for word sequences, expressed in terms of these representations. Generalization is obtained because a sequence of words that has never been seen before gets high probability if it is made of words that are similar (in the sense of having a nearby representation) to words forming an already seen sentence. Training such large models (with millions of parameters) within a reasonable time is itself a significant challenge. We report on experiments using neural networks for the probability function, showing on two text corpora that the proposed approach significantly improves on state-of-the-art n-gram models, and that the proposed approach allows to take advantage of longer contexts.

Neural Language Modeling

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*, 1137-1155.

1. Assign to each word in the vocabulary „a distributed feature vector"
   - We assume we have a vocabulary V
   - Each word $w \in V$ gets a d-dimensional vector $\mathbf{v}_w \in \mathbb{R}^d$
     - d usually much smaller than V
   - Vectors $\mathbf{v}_w$ of all $|V|$ vocabulary words stacked in a matrix

$$\mathbf{W} \in \mathbb{R}^{|V| \times d}$$

   - We will call $\mathbf{W}$ an embedding matrix of the LM
     - Encodes context-independent meanings of words

Neural Language Modeling

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*, 1137-1155.

2. Define language modeling probabilities as a function of vector representations (i.e., embeddings) of words

$$P(w_m | w_1 w_2 \ldots w_{m-1}) \approx g(\mathbf{v}_{m-n+1}, \ldots, \mathbf{v}_{m-2}, \ldots, \mathbf{v}_{m-1} | \boldsymbol{\omega})$$

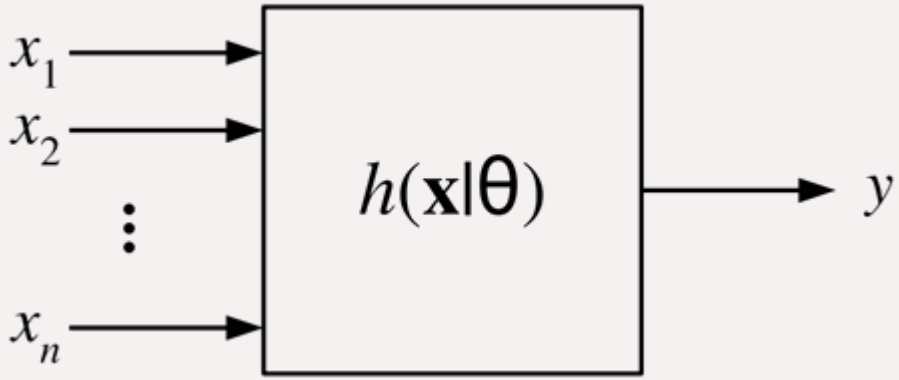
- „The function *g* may be implemented by a feed forward or recurrent neural network or another parametrized function, with parameters $\boldsymbol{\omega}$"

- The overall set of parameters of the model is $\boldsymbol{\theta} = (\mathbf{W}, \boldsymbol{\omega})$

Recap: (Supervised) Machine Learning

(Supervised) machine learning always has **three components**:

1. A model $h(\mathbf{x}|\boldsymbol{\theta})$: defines how the output is computed from input **x**
   - In deep learning models are highly parametrized compositions of non-linear functions (each individual function is a „layer")
     - **θ** – model's parameters



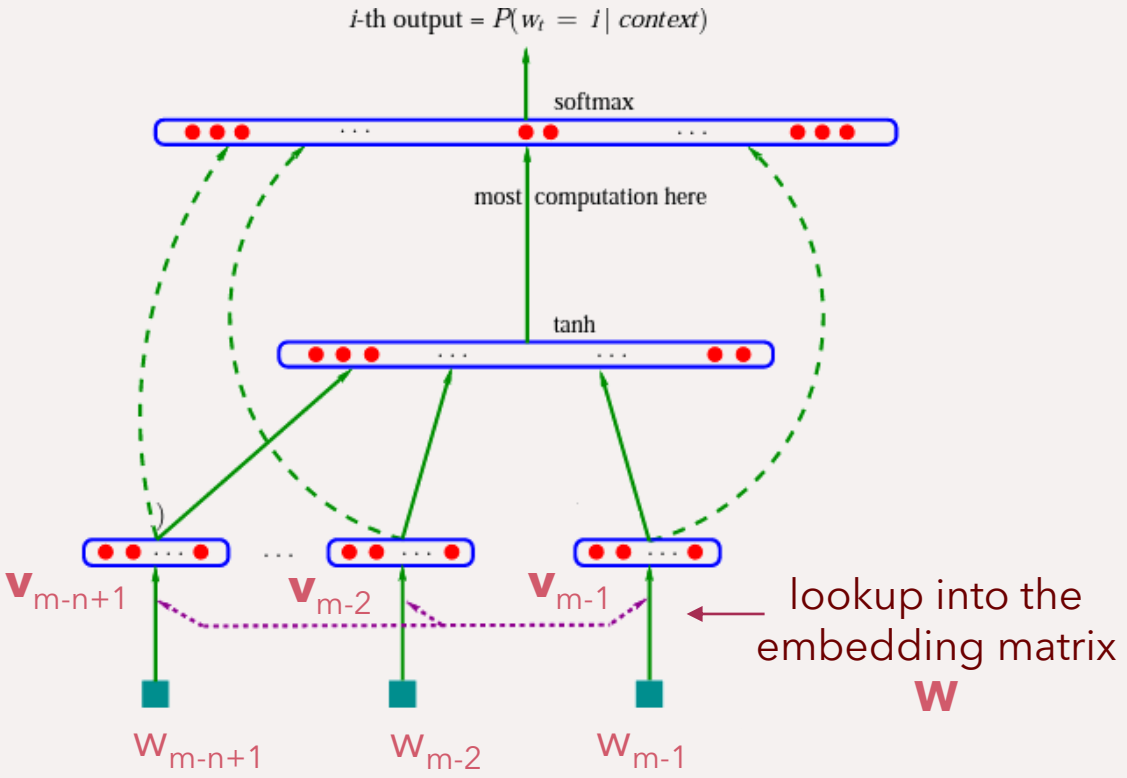
Neural Language Modeling

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*, 1137-1155.

- Input: concatenation of embeddings of context words

$\mathbf{x} = \mathbf{v}_{m-n+1} \oplus \ldots \mathbf{v}_{m-2} \oplus \mathbf{v}_{m-1}$

- $\mathbf{x}$ is of length $(n-1)d$



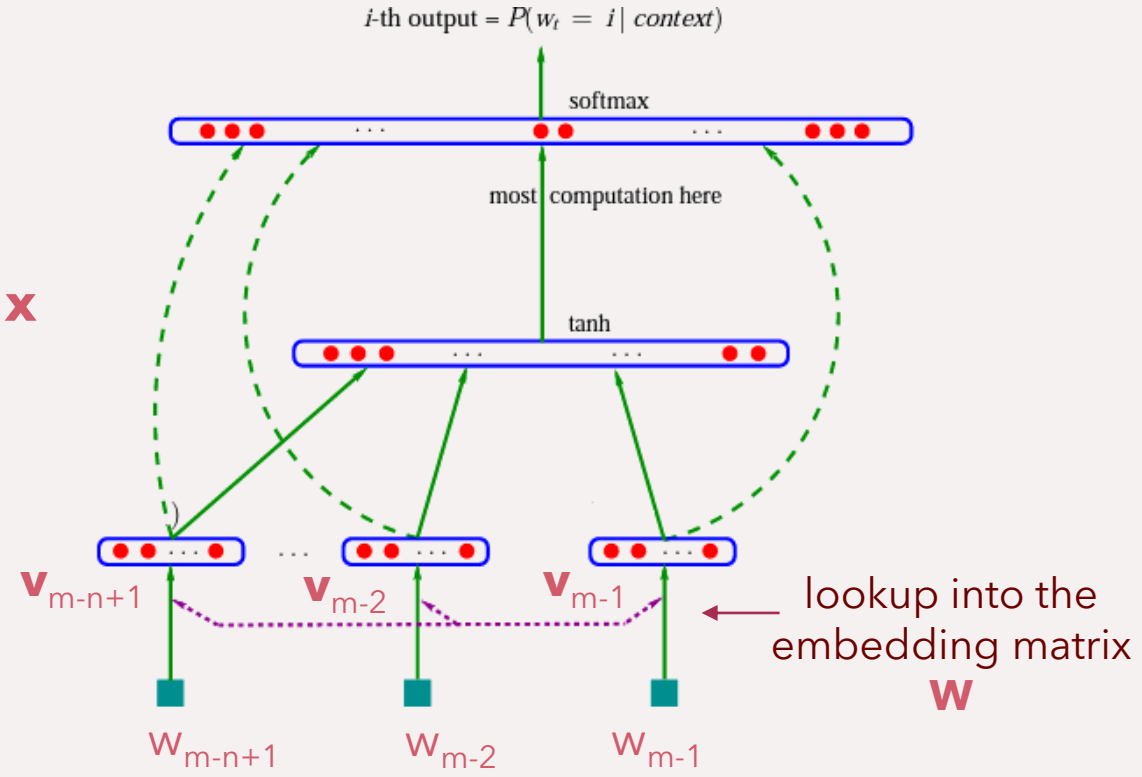
Neural Language Modeling

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*, 1137-1155.

- **Bengio's Neural LM**: The Model

$\hat{\mathbf{y}} = tanh(\mathbf{W_1 x} + \mathbf{b_1})\mathbf{W_2} + \mathbf{W_3 x} + \mathbf{b_2}$

  - Layer #1: non-linear down-projection of $\mathbf{x}$
    - $\mathbf{x}^{(1)} = tanh(\mathbf{W_1 x} + \mathbf{b_1})$
    - *tanh* = hyperbolic tangent
    - $\mathbf{W_1} \in \mathbb{R}^{h \times (n-1)d}$ and $\mathbf{b_1} \in \mathbb{R}^h$ parameters that downproject $\mathbf{x}$ from size $(n-1)d$ *to size* h

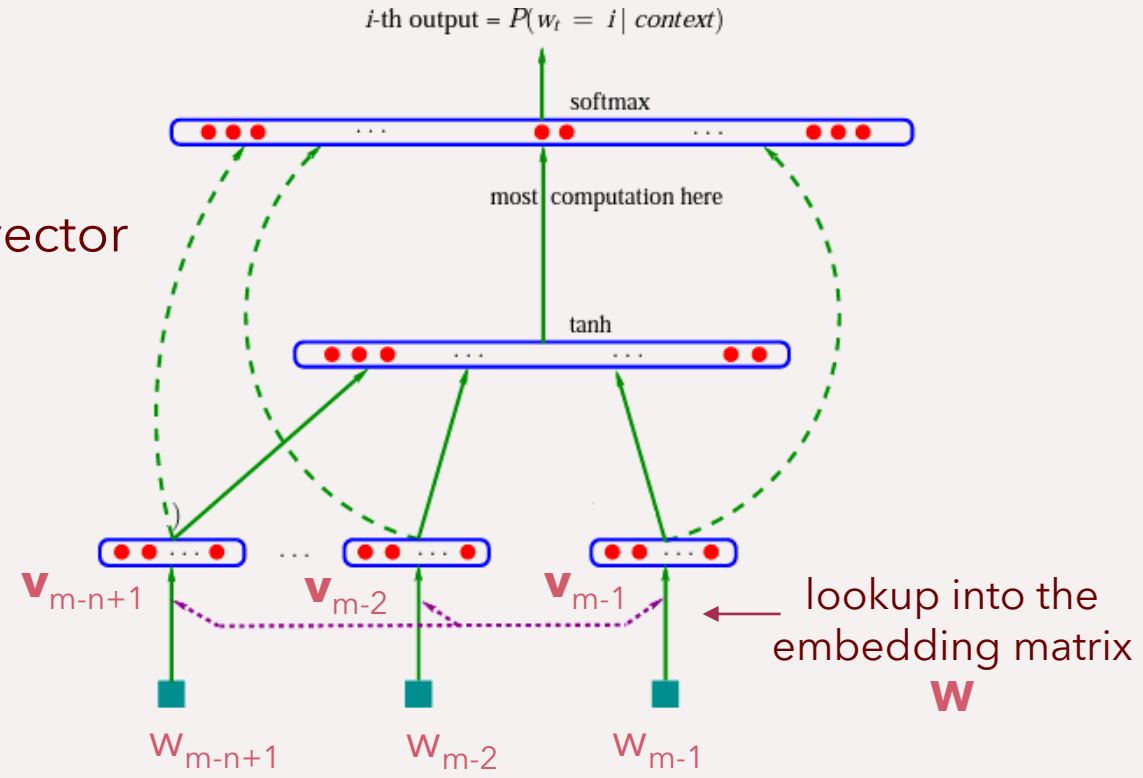


Neural Language Modeling

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*, 1137-1155.

$$\hat{\mathbf{y}} = \mathbf{W_2}\, tanh(\mathbf{W_1 x} + \mathbf{b_1}) + \mathbf{W_3 x} + \mathbf{b_2}$$

- Layer #2: linear projection of $\mathbf{x}^{(1)}$ into a vector of length $|V|$ (vocabulary size)
  - $\mathbf{x}^{(2)} = \mathbf{W_2 x}^{(1)} + \mathbf{b_2}$
  - $\mathbf{W_2} \in \mathbb{R}^{|V| \times h}$ and $\mathbf{b_2} \in \mathbb{R}^{|V|}$ parameters that up-project $\mathbf{x}^{(1)}$ from hidden size *h* to size $|V|$



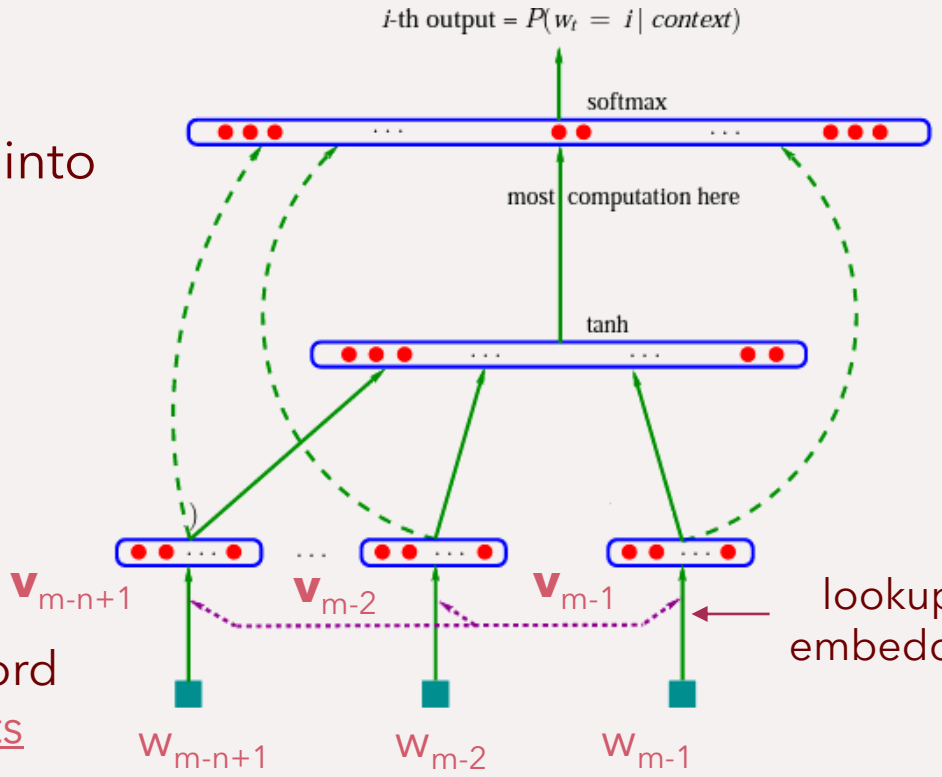
Neural Language Modeling

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*, 1137-1155.

$$\hat{\mathbf{y}} = \mathbf{W_2}\, tanh(\mathbf{W_1 x} + \mathbf{b_1}) + \mathbf{W_3 x} + \mathbf{b_2}$$

- Layer #3: parallel linear up-projection of **x** into a vector of length |V| (vocabulary size)
  - $\mathbf{x}^{(3)} = \mathbf{W_3 x}$
  - This we will call „residual connection"
  - $\mathbf{W_3} \in \mathbb{R}^{|V| \times (n-1)d}$

- Finally, $\hat{\mathbf{y}} = \mathbf{x}^{(2)} + \mathbf{x}^{(3)}$
  - Vector of |V| scores, one for each vocab. word
  - These unnormalized scores are called logits



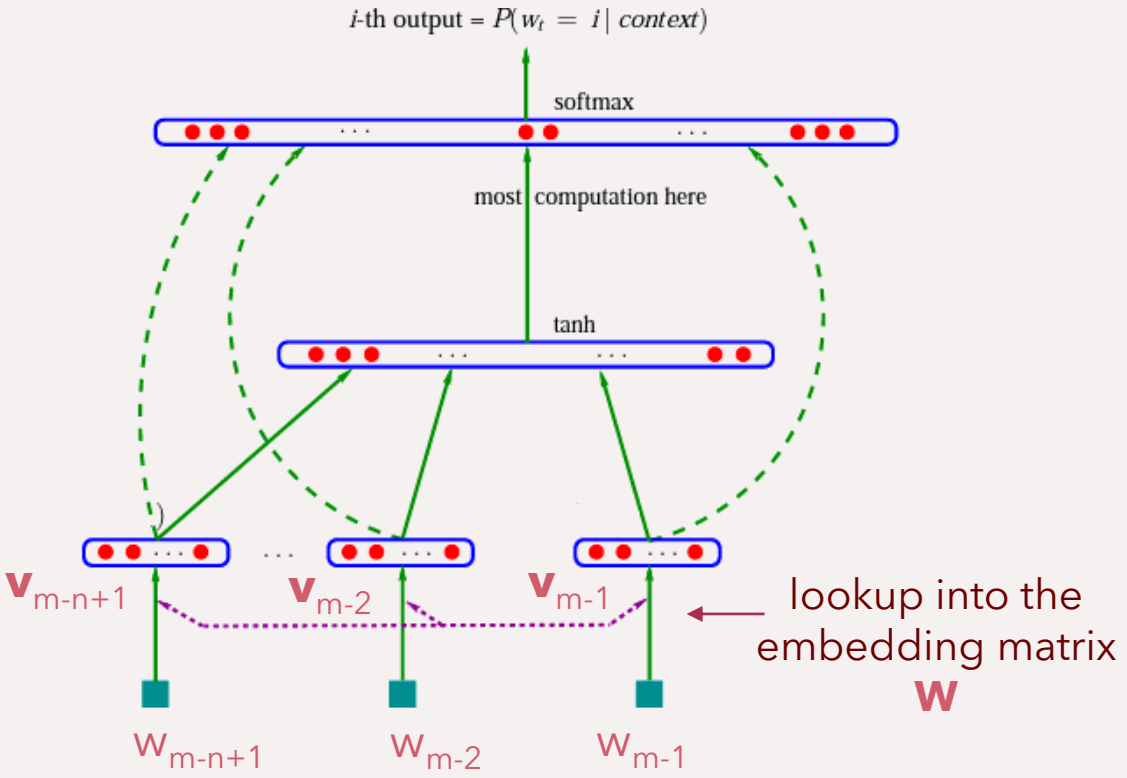
Neural Language Modeling

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*, 1137-1155.

$$\hat{\mathbf{y}} = \mathbf{W_2}\, tanh(\mathbf{W_1 x} + \mathbf{b_1}) + \mathbf{W_3 x} + \mathbf{b_2}$$

- $\hat{\mathbf{y}} \in \mathbb{R}^{|V|}$ is a vector of *logits*
- But we need $P(w \mid w_{m-n+1} \ldots w_{m-1})$ for each word w from the vocabulary V
- Need to convert $\hat{\mathbf{y}}$ into a probability distribution
- **Softmax** function:

$$\hat{y}_i \rightarrow \frac{e^{\hat{y}_i}}{\sum_{j=1}^{|V|} e^{\hat{y}_j}}$$



Recap: (Supervised) Machine Learning

(Supervised) machine learning always has **three components**:

1. Model
2. An objective function – quantifies „how correct" model's prediction prediction $h(\mathbf{x}|\boldsymbol{\theta})$ is w.r.t. to the desired output $\mathbf{y}$
   - Most commonly we minimize a loss function

Objective of **Bengio's LM**:

- For an observed sequence $w_{m-n+1}...w_{m-1}w_m$
- We want to maximize $P(w \mid w_{m-n+1}...w_{m-1})$ for the true word w_m
  - We want $P = 1$ for w_m and $P = 0$ for all other words from vocabulary V

- Common loss function in LMs: negative log-likelihood
  - $L(\mathbf{x}, \mathbf{y}|\boldsymbol{\theta}) = -\sum_{i=1}^{|V|} y_i \ln(\hat{y}_i)$ or $-\ln \hat{P}(w_m \mid w_{m-n+1}...w_{m-1})$

Recap: (Supervised) Machine Learning

(Supervised) machine learning always has **three components**:

1. Model
2. Objective function
3. Optimization algorithm – an algorithm that finds values $\hat{\boldsymbol{\theta}}$ for the model's parameters that optimize the objective function on the training dataset D = {($\mathbf{x}$, $\mathbf{y}$)}

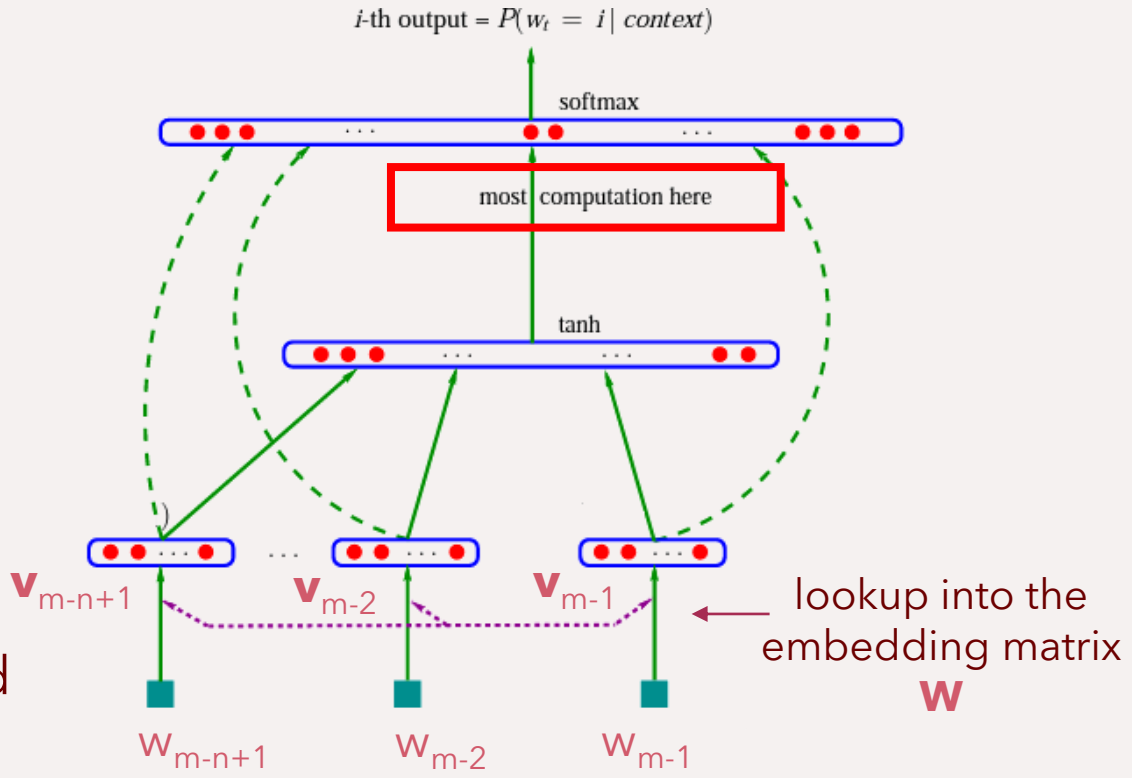
$$\hat{\boldsymbol{\theta}} = \text{argmin}_{\boldsymbol{\theta}} L(D|\boldsymbol{\theta})$$

- In deep learning: numerical optim. with gradient-based algorithms
- Layerwise, from last layer to the first – backpropagation (Lecture 3)

Neural Language Modeling

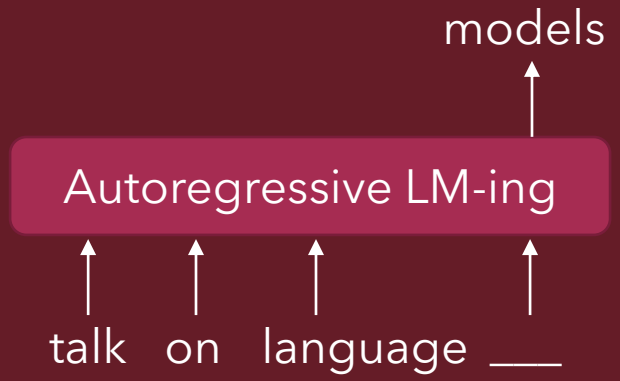
Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*, 1137-1155.

- Bengio's LM is from **2003**!!!
  - Today's LLMs are conceptually virtually identical!
- Q: why did it take 15-20 years for neural LMs to become „a thing"
- Short answer: hardware
  - softmax over large vectors is slow
  - Prevented Bengio's LM to be trained on (very) large text collections

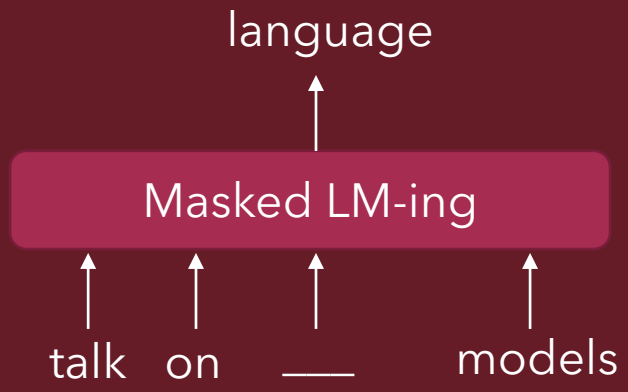


Types of Language Modeling

- Autoregressive (aka causal) LM-ing
  - Only preceding context available
  - Better for **lang. generation** tasks
  - Naturally, more of a „decoder"



- Masked LM-ing
  - Whole context available
  - Better for **lang. understanding** tasks
  - Naturally, more of an „encoder"



Neural Language Modeling

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

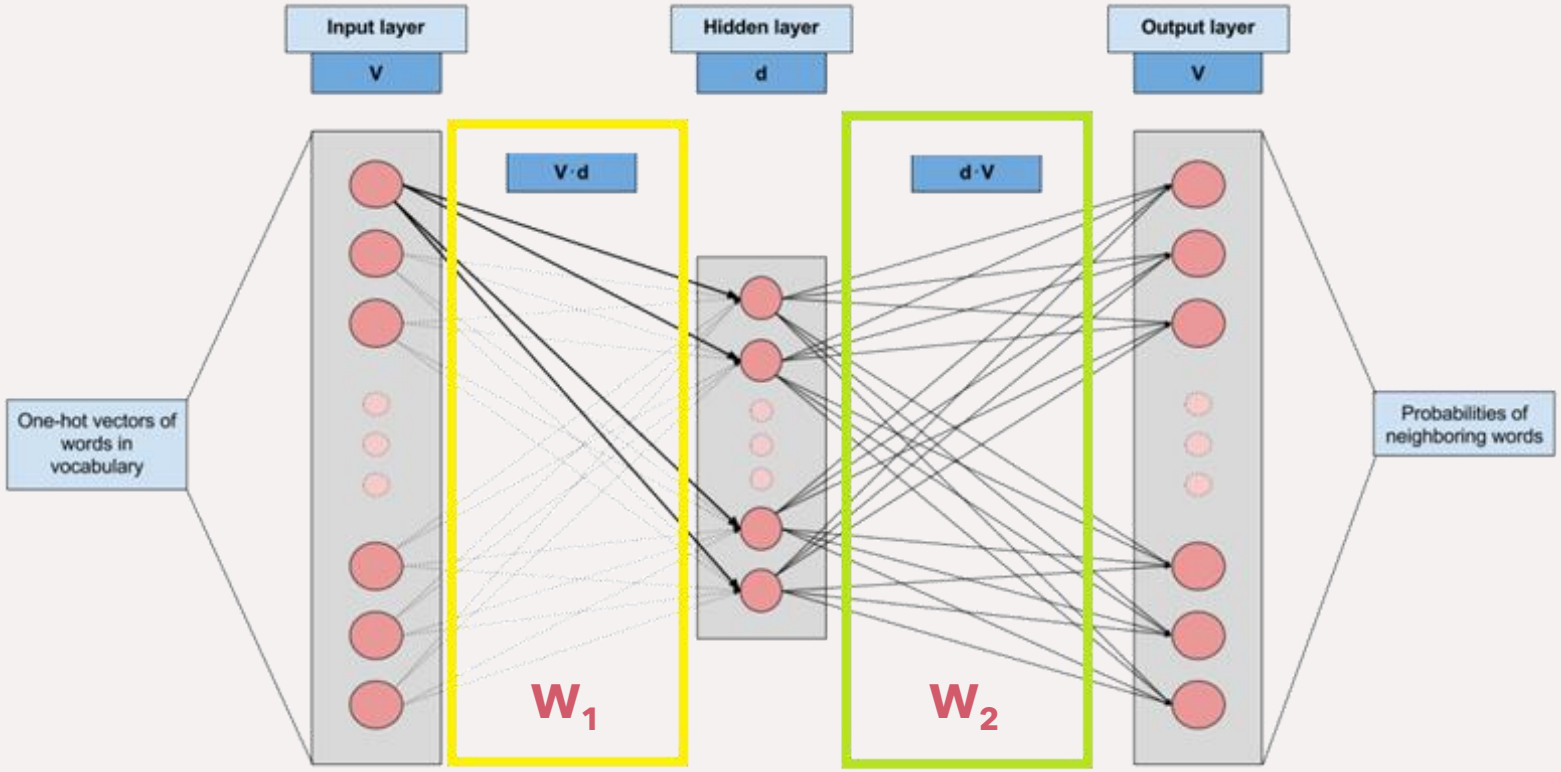
- Fast-forward 10 years: word embedding models
  - Two models: Skip-Gram and CBOW in a software package word2vec
- Shallower model than Bengio's LM
  - Word representations only parameters
  - Two vectors for each word from the vocabulary
    - Correspondingly, two embedding matrices

$$\mathbf{W_1} \in \mathbb{R}^{|V| \times d} \text{ and } \mathbf{W_2} \in \mathbb{R}^{d \times |V|}$$

- $\mathbf{v_1}(w)$: vector of word w in $\mathbf{W_1}$ (a row in the matrix)
- $\mathbf{v_2}(w)$: vector of word w in $\mathbf{W_2}$ (a column in the matrix)

Neural Language Modeling

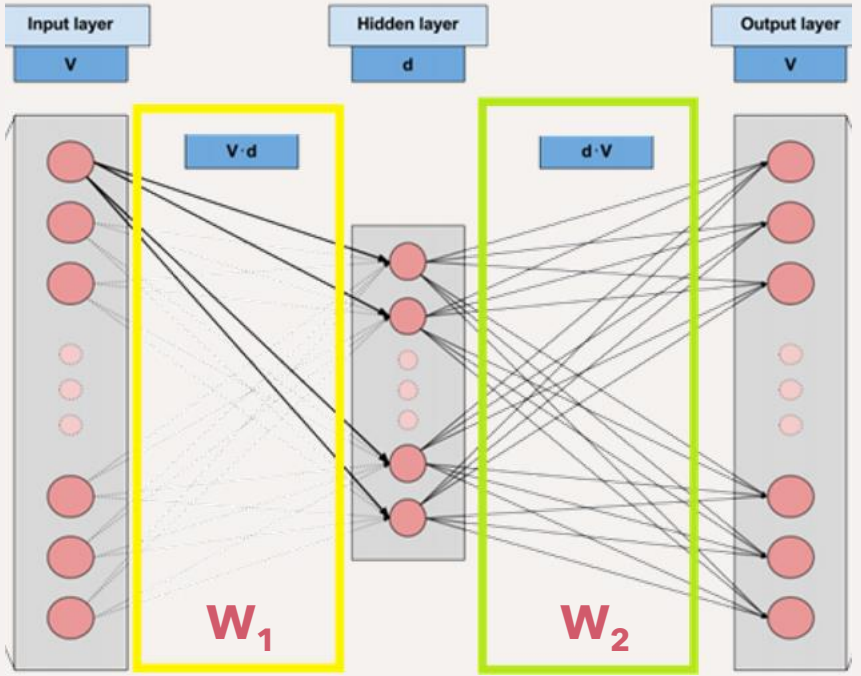
Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.



Neural Language Modeling

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

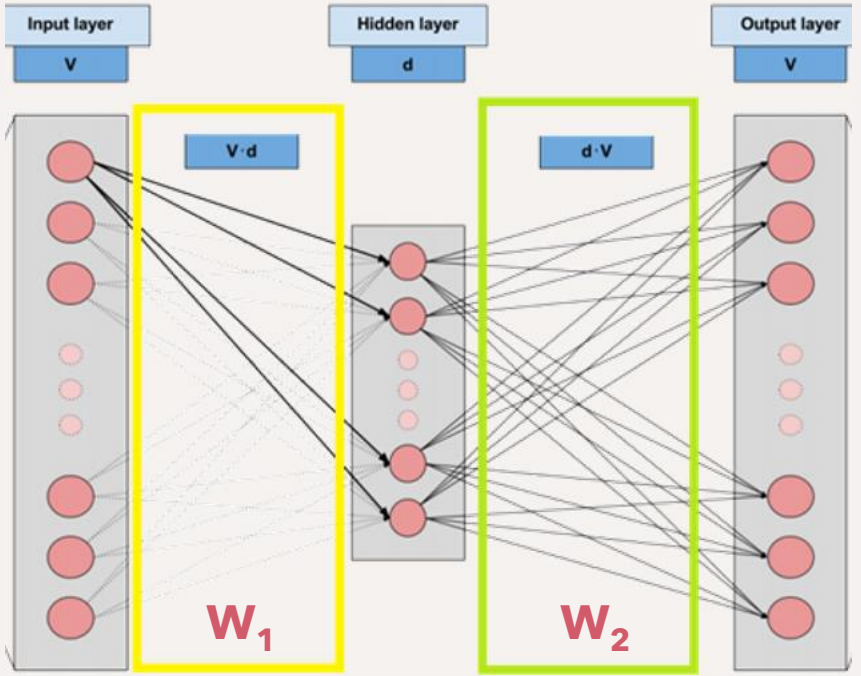
- $\mathbf{W_1} \in \mathbb{R}^{|V| \times d}$ and $\mathbf{W_2} \in \mathbb{R}^{d \times |V|}$
- Continous Bag-of-Words (CBOW)
  - predict central word from context
  - Effectively, masked LM-ing
  - Input: $w_{i-k} \ldots w_{i-1} [w_i] w_{i+1} \ldots w_{i+k}$
  - Model:
    - Layer #1: $\mathbf{v_c} = \frac{1}{2k} \sum_{j=-k}^{k} \mathbf{v}_1(w_{i+j})$
    - Layer #2: $\hat{\mathbf{y}} = \text{softmax}(\mathbf{v_c W_2})$
    - $\mathbf{y}$ = one-hot encoding of w_i



Neural Language Modeling

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

- $\mathbf{W_1} \in \mathbb{R}^{|V| \times d}$ and $\mathbf{W_2} \in \mathbb{R}^{d \times |V|}$
- Skip-Gram
  - Predict each of context words from the center word
  - One training instance is one pair w_i and w_{i+j} (j between $-k$ and k)
  - Model:
    - „Layer #1": just lookup – $\mathbf{v_c}$ is the row of $\mathbf{W_1}$ that corresponds to w_i
    - Layer #2: $\hat{\mathbf{y}} = \text{softmax}(\mathbf{v_c W_2})$
    - $\mathbf{y}$ = one-hot encoding of w_{i+j}



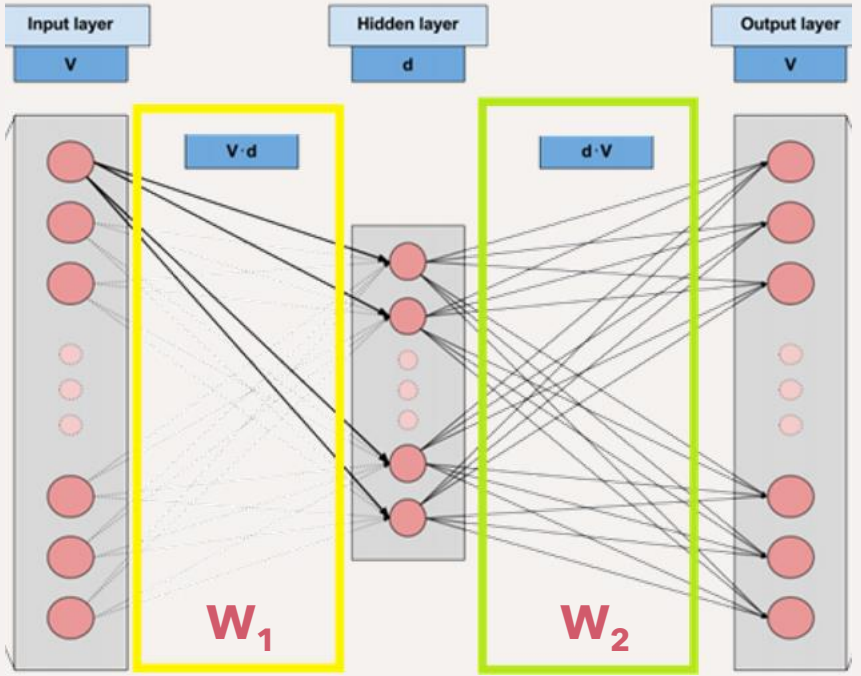
Neural Language Modeling

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

- Both Skip-Gram and CBOW:

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{v_c W_2})$$

- Lesson from Bengio: softmax over large vectors is slow
- Trick: **Negative sampling**
  - Multiply $\mathbf{v_c}$ only with a small subset of columns from $\mathbf{W_2}$
  - Must include the column of the „gold" word to be predicted („positive")...
  - ...and *N* randomly selected columns – „negatives"
  - Softmax over a vector of length *N*+1



Neural Language Modeling

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

VS.

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A Neural Probabilistic Language Model. *Journal of Machine Learning Research*, *3*, 1137-1155.

Q: Which model is more expressive/powerful?

- Skip-Gram/CBOW: produces only static word vectors
  - The vector of the word is always the same, regardless of the context
- Bengio's LM has more parameters than just word embeddings: these parameters essentialy contextualize word vectors against each other

Content

- What is Language Modeling?
- Neural Language Modeling

- **Tokenization**
  - **Word-Level vs. Character-Level**
  - Subwords

Tokenization

- So far, we always assumed a vocabulary V
  - What „words" should be in that vocabulary?
- In neural LM-ing, we typically build the vocabulary from a large training corpus on which we intend to train the LM
- Two key considerations (especially important for multilingual LMs):
  - Coverage
    - Minimize the number of tokens that are **unknown** (UNK) to the LM (i.e., unseen in the training corpus)
  - Memory
    - A very large vocabulary means a very large embedding matrix,
    - Neural LMs are trained on GPUs with limited VRAM

Word-level tokenization

- Whitespace tokenization: simply split the text on whitespaces
  - Some tweaks needed, e.g., for punctuation „ain't that funny."
  - Rule-based word-level tokenizers (typically language-specific)
- Issues: word-level tokenization is „sparse"
  - Languages that do not use whitespaces to delimit words
    - E.g., Mandarin Chinese (Simplified): 今天是维尔茨堡的一个好日子
    - E.g., Komposita in German: „Rhabarberbarbarabarbarbaren"
  - Word-level tokenization doesn't reflect morphosyntax of the language
    - *tokenization vs. token*
    - *tokenization vs. industrialization*
  - Word-level tokenization results in very large vocabularies
    - Nonetheless, any word not seen in training data is UNK

Character-level tokenization

- How about each character being its own token
  - Through LM-ing, we learn an embedding for each character

- Advantages
  - Small vocabulary
  - Even if we collect all characters from all scripts in the world, still merely a few thousand symbols
  - Simple and super-fast!

Character-level tokenization

- How about each character being its own token
  - Through LM-ing, we learn an embedding for each character
- Shortcomings
  - Embedding vectors supposed to encode context-independent meaning of tokens
    - Characters have no intrinsic meaning* (recall: morphemes)
    - What's the context-independent meaning of „a"?
  - Hard to obtain meaningful representations for units that do have meaning – morphemes or words – from character embeddings
    - Character-level tokenization commonly leads to performance loss
  - Texts/sentences (e.g., in classification) very long sequences of character embeddings – only short texts would fit into GPU memory

Content

- What is Language Modeling?
- Neural Language Modeling

- **Tokenization**
  - Word-Level vs. Character-Level
  - **Subwords**

Subword tokenization

- Sweet spot between word- and character-level tokenization
  - Searching for the optimal tradeoff between
    - Memory footprint of an LM and
    - LM's ability to learn semantically meaningful text representations
- Subword tokenization is:
  - More memory-efficient than word-level tokenization
  - More semantically meaningful than character-level tokenization

Subword tokenization

- What we know from early work on LMs (Bengio, Mikolov):
  - Token frequency correlates with embedding quality

- Core ideas of **subword tokenization**:
  1. Frequent words should not be split into smaller parts
  2. Less frequent words split into subwords that occur more frequently

  - Example: „token" vs. „tokenization"
    - „tokenization" split into subwords „token" and „ization"
    - both „token" and „ization" will have larger corpus frequency

  - Q: how to decide (1) what to split and (2) how to split it?

Subword tokenization

- What we know from early work on LMs (Bengio, Mikolov):
  - Token frequency correlates with embedding quality
- Q: how to decide (1) what to split and (2) how to split it?
- Different subword tokenization algorithms
  - Byte-Pair Encoding (BPE)
  - WordPiece
  - SentPiece

Byte-Pair Encoding

Sennrich, R., Haddow, B., & Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (pp. 1715-1725).

- Introduced in the context of neural machine translation (NMT), before pretraining neural (L)LMs was „a thing"
- Requires a (typically language-specific) pre-tokenizer
  - E.g., whitespace or rule-based tokenizer
- Result of pre-tokenization of the training corpus:
  - A set of word-level tokens with occurrence frequencies

Byte-Pair Encoding

Sennrich, R., Haddow, B., & Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (pp. 1715-1725).

- Step 1: pre-tokenize and produce the set of word-level tokens V_w

  Toy example: V_w = {son:8, ton: 4, top:12, pop:3, sons:4}

- Step 2: build the initial **base vocabulary**
  - Initial base vocabulary: all characters in words from V_w

    V_B = {n, o, p, s, t}

  - V_w (as per V_B): {s, o, n: 8; t, o, n: 4; t, o, p: 12; p, o, p: 3; s, o, n, s: 4}

Byte-Pair Encoding

Sennrich, R., Haddow, B., & Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (pp. 1715-1725).

- Step 3 (repeat until desired vocabulary size is reached):
  - Count the frequency of each pair of „tokens" from V_B
  - Merge the two tokens V_B with highest frequency of co-occurrence across the words from V_w

V_w = {s, o, n: 9; t, o, n: 4; t, o, p: 12; p, o, p: 3; s, o, n, s: 4}
V_B = {n, o, p, s, t} (desired vocab size: 8 tokens)

Iteration #1: „o"+„n" have the highest frequency of 17 („son", ton", and „sons")
- Merge „o" and „n" into „on" in all corresponding words from V_w

Byte-Pair Encoding

Sennrich, R., Haddow, B., & Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (pp. 1715-1725).

- Step 3 (repeated until desired vocabulary size is reached):
  - Count the frequency of each pair of „tokens" from V_B
  - Merge the two tokens V_B with highest frequency of occurrence across the words from V_w

V_w = {s, on: 9; t, on: 4; t, o, p: 12; p, o, p: 3; s, on, s: 4}
V_B = {n, o, p, s, t, on} (desired vocab size: 8 tokens)

Iteration #2: „o"+„p" have the highest frequency of 15 („top", and „pop")

- Merge „o" and „p" into „op" in all corresponding words from V_w

Byte-Pair Encoding

Sennrich, R., Haddow, B., & Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (pp. 1715-1725).

- Step 3 (repeated until desired vocabulary size is reached):
  - Count the frequency of each pair of „tokens" from V_B
  - Merge the two tokens V_B with highest frequency of occurrence across the words from V_w

V_w = {s, on: 9; t, on: 4; t, op: 12; p, op: 3; s, on, s: 4}

V_B = {n, o, p, s, t, on, op}

Iteration #3: „s"+„on" have the highest frequency of 13 („son", and „sons")

- Merge „s" and „on" into „son" in all corresponding words from V_w

Byte-Pair Encoding

Sennrich, R., Haddow, B., & Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (pp. 1715-1725).

- Step 3 (repeated until desired vocabulary size is reached):
  - Count the frequency of each pair of „tokens" from V_B
  - Merge the two tokens V_B with highest frequency of occurrence across the words from V_w

$$V_w = \{son: 9; t, on: 4; t, op: 12; p, op: 3; son, s: 4\}$$
$$V_B = \{n, o, p, s, t, on, op, son\}$$

  - We reached the desired vocabulary size of 8 tokens
  - Merge rules we obtained: 1. o+n -> on, 2. o+p -> op, 3. s+on -> son

Byte-Pair Encoding

Sennrich, R., Haddow, B., & Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (pp. 1715-1725).

- **Inference:** applying the tokenizer once we obtained the vocabulary V_B
  - Closely mirrors the training process

Step 1: apply the same pre-tokenizer on the new text
E.g., „no nop sonop" → {no; nop; sonop}

Step 2: start from individual characters for each word-level token
- keep merging using the merge rules learned in training

n, o → no rule applies, separate into two tokens [n, o]

Merge rules
1. o+n -> on
2. o+p -> op
3. s+on -> son

Byte-Pair Encoding

Sennrich, R., Haddow, B., & Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (pp. 1715-1725).

- **Inference:** applying the tokenizer once we obtained the vocabulary V_B
  - Closely mirrors the training process

Step 1: apply the same pre-tokenizer on the new text
E.g., „no nop sonop" → {no; nop; opt}

Step 2: start from individual characters for each word-level token
- keep merging using the merge rules learning in training

n, o, p → 2. merge rule applies, separate into two tokens [n, op]

Merge rules
1. o+n -> on
2. o+p -> op
3. s+on -> son

Byte-Pair Encoding

Sennrich, R., Haddow, B., & Birch, A. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers) (pp. 1715-1725).

- **Inference:** applying the tokenizer once we obtained the vocabulary V_B
  - Closely mirrors the training process

Step 1: apply the same pre-tokenizer on the new text
E.g., „no nop sonop" → {no; nop; opt}

Step 2: start from individual characters for each word-level token
- keep merging using the merge rules learning in training

s, o, n, o, p → all three rules apply (in order of rules), separate into two tokens [son, op]

Merge rules
1. o+n -> on
2. o+p -> op
3. s+on -> son

WordPiece Tokenization

Devlin, J., Chang, M., Lee, K., & Toutanova, L. K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

- Tokenization algorithm proposed with BERT
  - Training similar to BPE, with two main differences:

1. Initial vocabulary base V_B distinguishes characters that start words from those that are inside of the word-level tokens

Toy example: V_w = {son:8, ton: 4, top:12, pop:3, sons:4}
V_B = {p, s, t, ##o, ##n, ##p, ##s} (## prefix for „inside" chars)

WordPiece Tokenization

Kenton, J. D. M. W. C., & Toutanova, L. K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

- Tokenization algorithm proposed with BERT
  - Training similar to BPE, two main differences:

2. Merge of tokens based on relative frequency score

$$\frac{f(t_1 t_2)}{f(t_1) f(t_2)}$$

  - In essence very similar to pointwise mutual information (PMI)
  - Prioritizes merging of pairs with lower-frequency parts
    - E.g., „like" and „##ly" will not be merged early

WordPiece Tokenization

Kenton, J. D. M. W. C., & Toutanova, L. K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of NAACL-HLT (pp. 4171-4186).

- Tokenization algorithm proposed with BERT
- **Inference** is very different from BPE
  - WordPiece keeps only the final vocabulary V_B
    - Not the merge rules
  - Greedy splitting: find the longest substring that is in V_B
    - Split on it and repeat for the remainder of the word-level token
    - E.g., {transform, ##ers, trans, ##form, transformer, ##s} all in V_B
      - Q: how will WordPiece tokenize „transformers"?

SentencePiece Tokenization

Kudo, T., & Richardson, J. (2018, November). SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In Proceedings of EMNLP Processing: System Demonstrations (pp. 66-71).

- BPE and WordPiece rely on a pre-tokenizer and its word tokenization
  - Word-level tokenization is, in principle, language-specific
  - Some languages don't have „words" as such
  - Problem for multilingual LMs
- SentencePiece
  - Does not require a pre-tokenizer, creates V_B from raw sentences
    - It does need sentence segmentation, but this is more consistent across languages

SentencePiece Tokenization

Kudo, T., & Richardson, J. (2018, November). SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. In Proceedings of EMNLP Processing: System Demonstrations (pp. 66-71).

- **SentencePiece** is essentially BPE, **but**
  - Whitespace (denoted with underscore „_") is a character-level token, like any other character
  - Each sentence in the corpus is now for SentencePiece what a „word-level" token was for BPE
    - „today is my day" → [t, o, d, a, y, _, i, s, _, m, y, _, d, a, y]

Image: Alexander Mikhalchyk