ETH zürich

Language Modeling with *n*-grams and RNNs

Ryan Cotterell, Niklas Stoehr and Lucas Torroba Hennigen

Administrivia

Project and Course Assignment Update

- **Course Research Project:** guidelines for the research-based course assignment have been released: https://rycolab.github.io/classes/intro-nlp/materials/project_guidelines.pdf

- **Course Assignment:** we are actively working on the individual course assignment
  - There are going to be some good questions for you to answer!
    - The questions will cover Lectures 1–9
  - We hope to get the final version out by November 1st, 2020
    - Why so long? We need to vet the questions with other students (not in the class), so we can assign points to them and make sure everything is doable in the proper amount of time

What’s Difference About This Lecture?

- We made fewer slides (about 60) instead of the normal amount (about 80)
- This should make a more drawn-out lecture that helps more people absorb the material
- **Backup Plan**: I give a “whiteboard lecture” on the a tablet if we have a lot of extra time

Structure of this Lecture

1 Structured Prediction — 2 *n*-gram models — 3 Recurrent Neural Networks

1. Structured Prediction
   - Kleene Closure
   - Prefix Tree
   - Local Normalization
2. *n*-gram models
   - *n*-gram Language Modelling
   - Log-Linear and MLP
   - Linguistic Limitations
3. Recurrent Neural Networks
   - Elman networks (Vanilla RNN)
   - GRUs and LSTMs
   - Bias-Variance Tradeoff
   - So what is Smoothing?

Supplementary Material

Reading: Eisenstein Ch. 6 and Ch. 9; Goodfellow, Bengio and Courville Ch. 10
Supplementary Reading: Good Tutorial on *n*-gram smoothing, Good–Turing Smoothing, Kneser and Ney (1995), Bengio et al. (2003), Mikolov et al. (2010)

Structured Prediction

1 What is Structured Prediction?

- **Last two lectures:** Modeling distributions over “small” sets. We implicitly assumed that we could easily sum over $\mathcal{Y}$

- **Next few lectures (including this one!):** Modeling more interesting structured objects

1 What is Structured Prediction?

Big picture: It’s just multi-class classification combined with an algorithm of our choosing!

In more detail:

- A class of problems that involves predicting structured objects (e.g., strings, trees or graphs) rather than scalar discrete or real values (e.g., sentiment class)
- Structured prediction is the intersection of algorithms and high-dimensional statistics
- The output space of the prediction problem is exponentially large and so we must now think carefully about how to evaluate outputs

1 What is Structured Prediction?

Recap: How have we been approaching modeling in this course?

- Modelling the probability distribution over outputs given inputs

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{\exp\{\text{score}(\mathbf{y},\mathbf{x})\}}{\sum_{\mathbf{y}' \in \mathcal{Y}} \exp\{\text{score}(\mathbf{y}',\mathbf{x})\}}$$

- **Question**: What if $|\mathcal{Y}|$ is really, really big? Can we find an efficient algorithm for computing that sum in the denominator?

① What is Structured Prediction?

Recap: How have we been approaching modeling in this course?

- Modelling the probability distribution over outputs given inputs

function of our choosing...

$$p(\mathbf{y} \mid \mathbf{x}) = \left.\frac{\exp\{\overset{\nwarrow}{\text{score}}(\mathbf{y}, \mathbf{x})\}}{\sum_{\mathbf{y}' \in \mathcal{Y}} \exp\{\text{score}(\mathbf{y}', \mathbf{x})\}}\right\} \text{softmax}$$

- **Question**: What if $|\mathcal{Y}|$ is really, really big? Can we find an efficient algorithm for computing that sum in the denominator?

① What is Structured Prediction?

Consider the size of the output space of various NLP tasks...

$$|\mathcal{Y}| = 2$$

Sentiment Analysis:
Is sentiment positive or negative?

$$|\mathcal{Y}| = n$$

Movie Genre Prediction:
Which genre is this script?

$$|\mathcal{Y}| = 2^n$$

Part-of-Speech Tagging:
This sentence has which part-of-speech-tag sequence?

1 What is Structured Prediction?

Consider the size of the output space of various NLP tasks...

$|\mathcal{Y}| = 2$

Sentiment Analysis:
Is sentiment positive or negative?

Bad idea to brute force solve this...

$|\mathcal{Y}| = n$

Movie Genre Prediction:
Which genre is this script?

$|\mathcal{Y}| = 2^n$

Part-of-Speech Tagging:
This sentence has which part-of-speech-tag sequence?

① What is Structured Prediction?

Bottom Line: We need an additional set of tools when our output space is exponential

$$\textbf{Statistics} \cap \textbf{(Theoretical) Computer Science}$$

We'll get to this in next class! $\nearrow$

ETH zürich

1 Language Modeling is Structured Prediction

- We have a finite set of words *V*, which we call our vocabulary, e.g. *V* = {a, b, c}

- The task is modeling the **distribution over sequences from V^***
  - The * marks the Kleene closure (explained on next slide!)

- **For instance:** BOS → a → a → b → a → c → a → EOS
  BOS → EOS
  BOS → c → a → EOS
  ….
- **Note:** BOS is a token indicating the “beginning of a sentence”
  EOS means “ending of a sentence”

1 What is Kleene Closure

- All possible outputs are within a set referred to as the **Kleene closure**

- V^* = {BOS, EOS, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, ...}

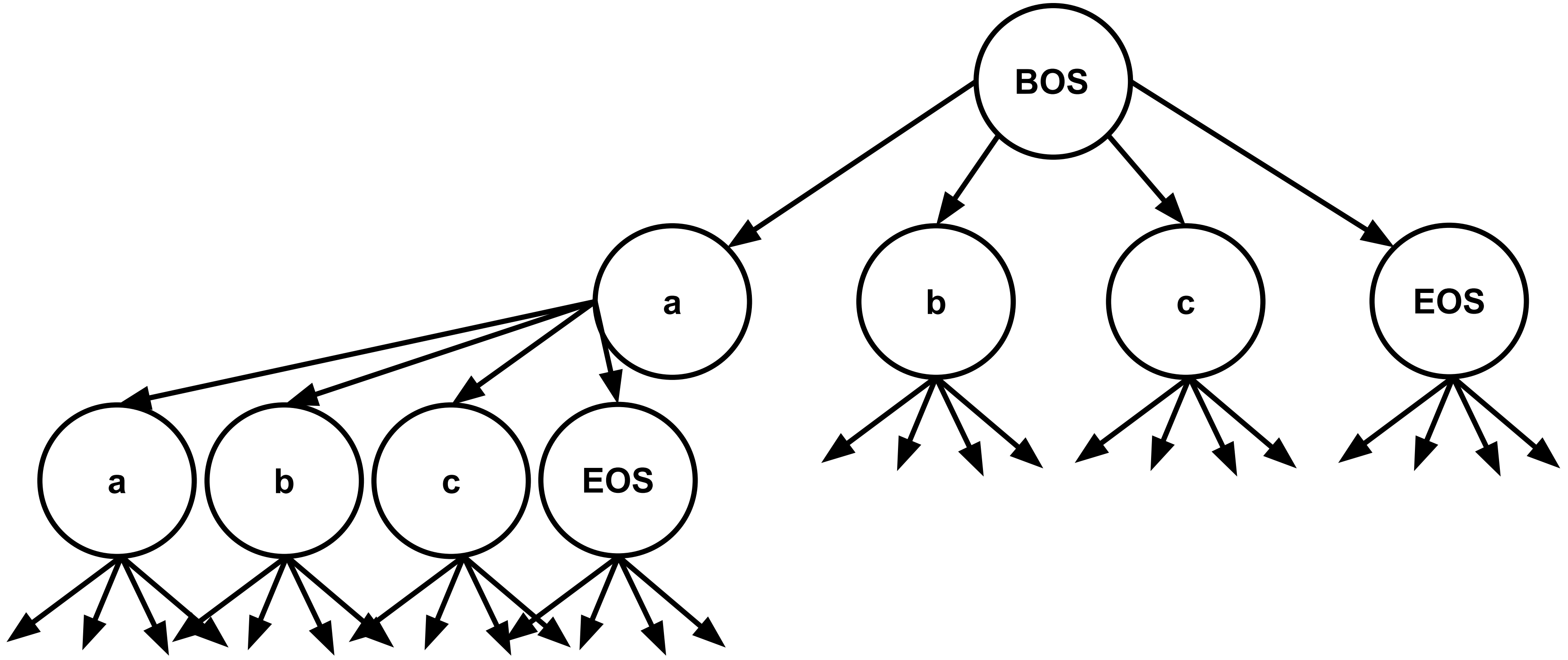
- What is the size of V^* ?
  - If we do not want to impose any prior assumptions, $|V^*|$ is **infinitely large**
  - Even if we bounded the length sequences of length at most n, we have $|V^{\leq n}|$, which is **still huge**

- How do we pronounce Kleene? Kleene pronounce it [kleɪni], but most computer scientists say [kliːni]

ETH zürich

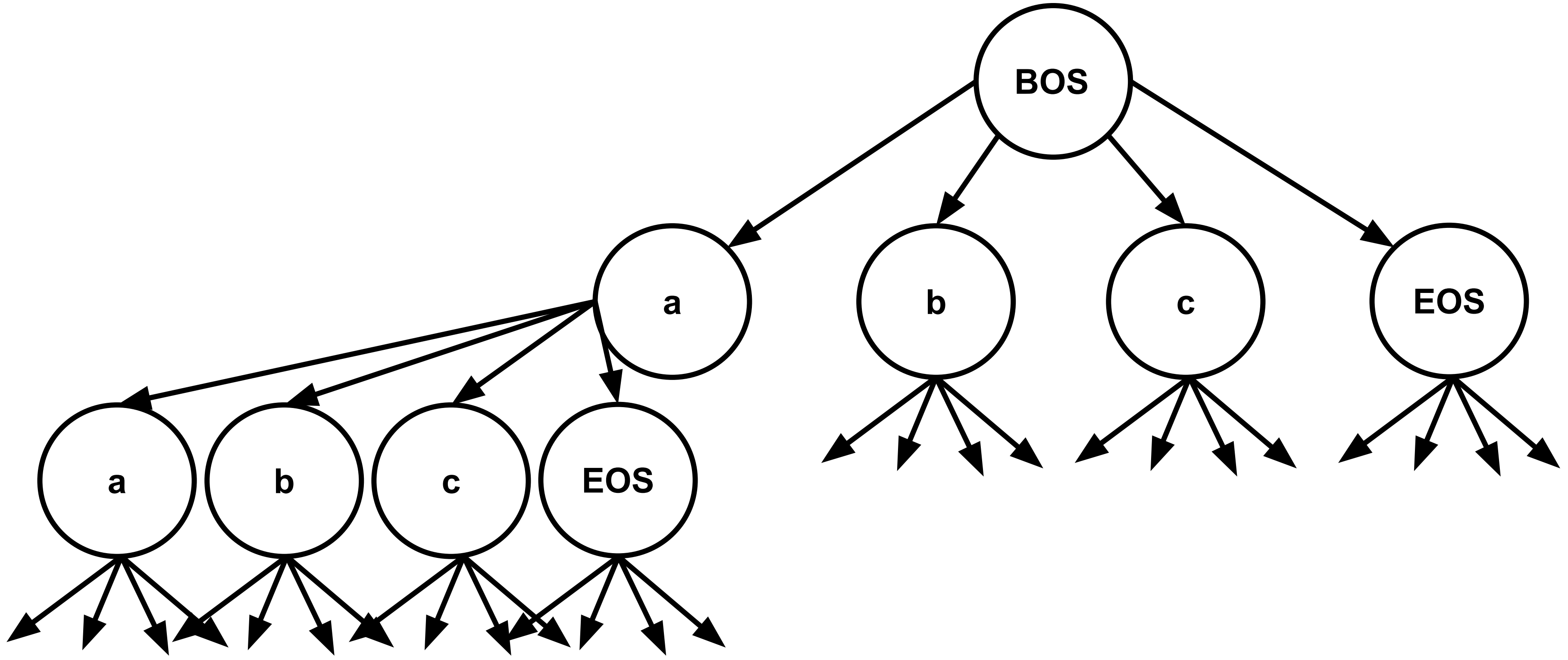
1 Prefix Tree

- The Kleene closure can thought of as a **prefix tree**



… where there is an infinite number of paths, each of which should have ***finite*** length

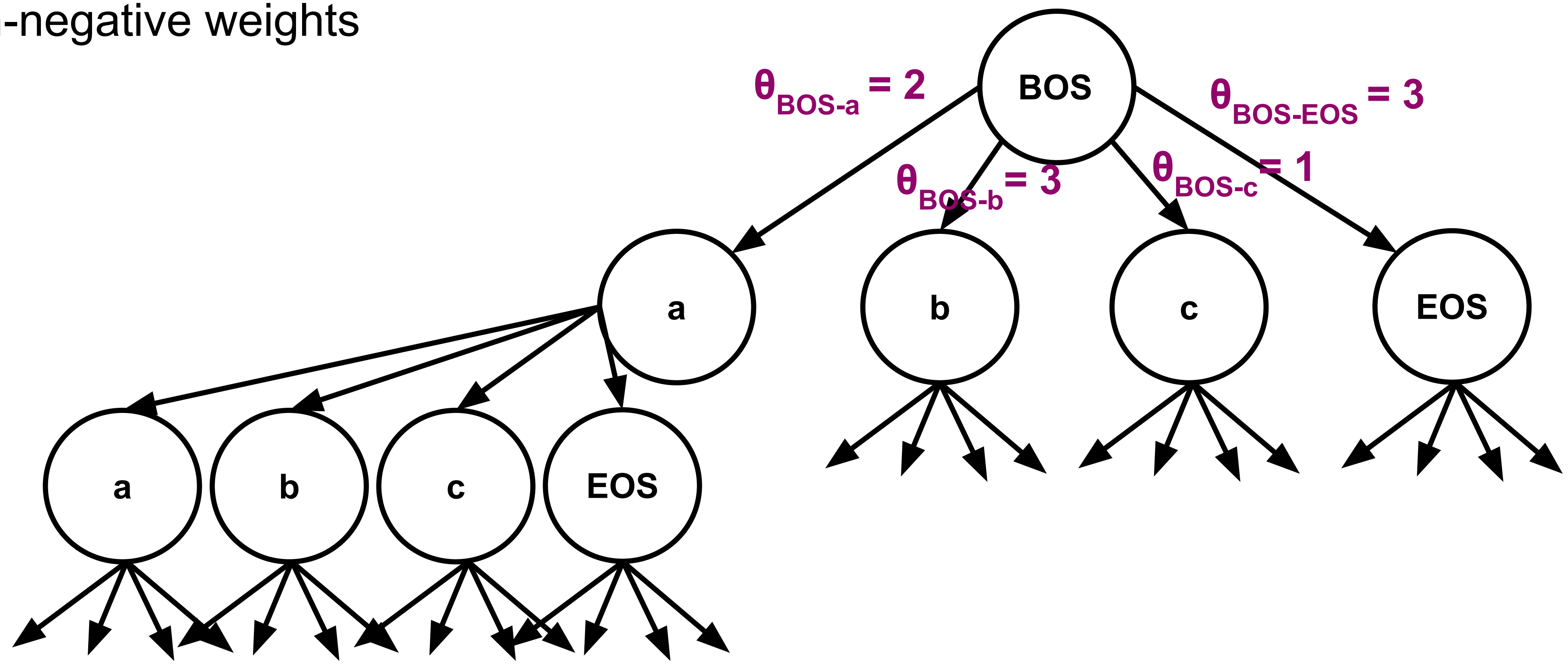
1 Prefix Tree and Language Modelling



→ a language model is the weighting of the prefix tree

1 Prefix Tree and Language Modelling

- The weighting may arbitrary non-negative weights



$\theta_{BOS\text{-}a} = 2$
$\theta_{BOS\text{-}b} = 3$
$\theta_{BOS\text{-}c} = 1$
$\theta_{BOS\text{-}EOS} = 3$
BOS
a
b
c
EOS
a
b
c
EOS

→ a language model is the weighting of the prefix tree

① What is a Language Model?

Almost: We also need to worry about normalization!

1 The Big Question: How do we Normalize the Language Model?

- *V** = {BOS, EOS, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, ...}

→ Our goal is to construct a **distribution over Kleene closure of a set**

- $p(\mathbf{y})$ where $\mathbf{y} \in V^*$

T is the length of **y**

θ are the weights on the prefix tree

- $p(\mathbf{y}) = \frac{1}{Z} \prod_{t=1}^{|\mathbf{y}|} \theta_{\mathbf{y}_{\leq t}}$ $\Rightarrow$ $Z = \sum_{\mathbf{y}' \in V^*} \prod_{t=1}^{|\mathbf{y}'|} \theta_{\mathbf{y}'_{\leq t}}$

  where Z is the normalising constant

- $\leq t$ as a subscript denotes the first *n* elements of string $\mathbf{y}_{\leq t}$

ETH zürich

1 Normalization in Structured Prediction

Problem: How do we compute the normalizing constant *Z*, which is the sum over an infinite set?

$$Z = \sum_{\mathbf{y}' \in V^*} \prod_{t=1}^{|\mathbf{y}'|} \theta_{\mathbf{y}'_{\leq t}}$$

Two Approaches:

Global Normalization | **Local Normalization**

1 Normalization in Structured Prediction

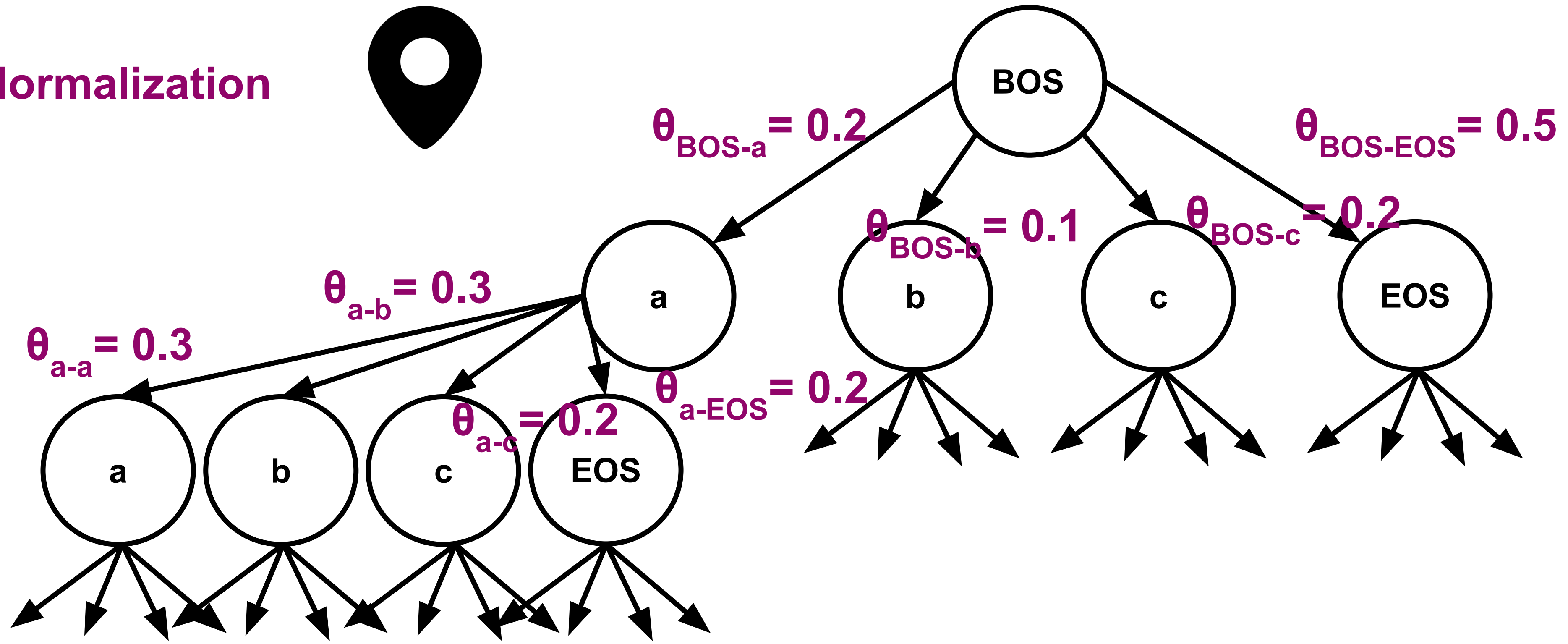
Global Normalization

- We need to come up with an efficient algorithm to compute Z
- This will be content of lecture #6, #7 and #8
- In the case of language modeling, there are no general-purpose algorithms for computing Z
    - In the case of certain backoff language models, we can run Floyd–Warshall which is cubic in the number of contexts
    - We don't have time to cover this wonderful algorithm (whose generalized is also due to Kleene!) in this class

ETH zürich

1 Local Normalization in Language Modeling

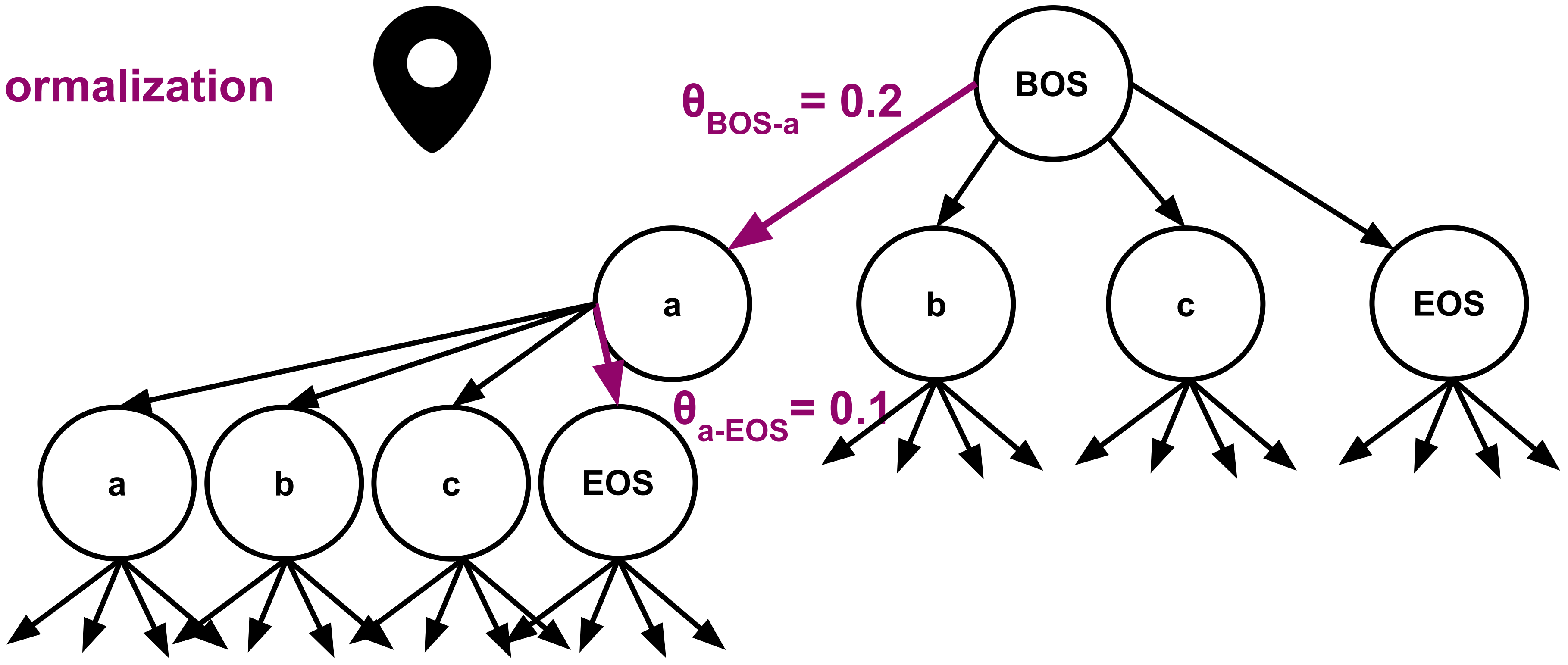
Local Normalization


BOS
$\theta_{BOS\text{-}a} = 0.2$
$\theta_{BOS\text{-}b} = 0.1$
$\theta_{BOS\text{-}c} = 0.2$
$\theta_{BOS\text{-}EOS} = 0.5$
a
b
c
EOS
$\theta_{a\text{-}a} = 0.3$
$\theta_{a\text{-}b} = 0.3$
$\theta_{a\text{-}c} = 0.2$
$\theta_{a\text{-}EOS} = 0.2$
a
b
c
EOS

Choose the weights strategically θ such that Z = 1, i.e. $Z = \sum_{\mathbf{y}' \in V^*} \prod_{t=1}^{|\mathbf{y}'|} = 1$

1 Local Normalization in Language Modeling

Local Normalization

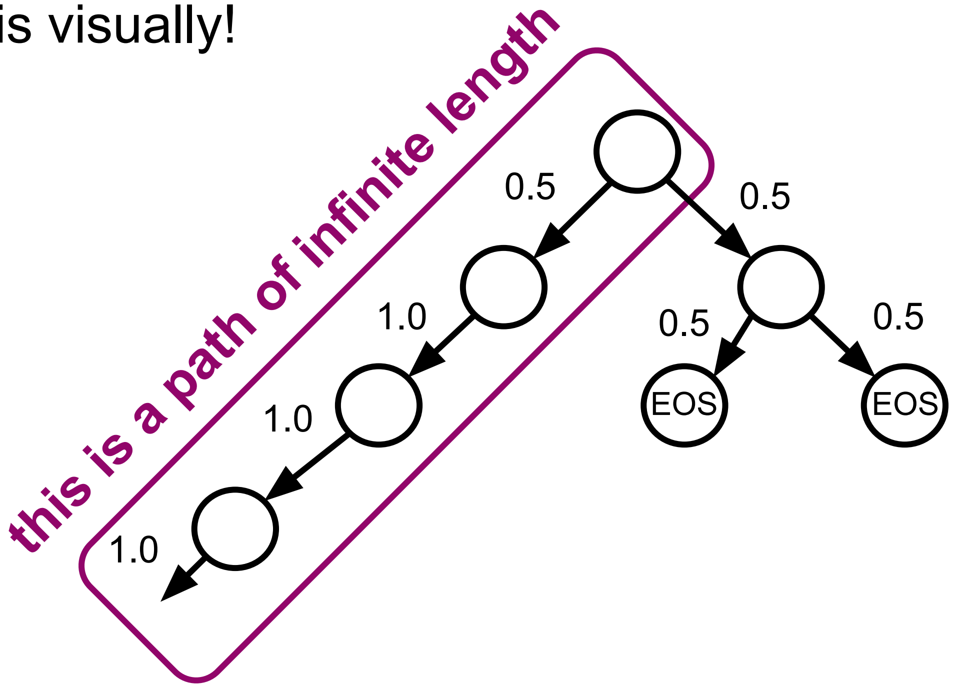


- Local Normalization provably guarantees that the model's normalizing constant Z is 1
- **Remember:** you multiply the weights along a path, then sum all paths
- **Definition:** the probability of all children given their parents is 1

1 Normalization in Structured Prediction

Local Normalization

- **Necessary Condition:** In a well defined prefix tree, every node has "EOS" as a descendant. This ensures we have an infinite number of paths with ***finite*** length
- What if the condition is not met? Then, we have ***deficient*** distributions, i.e. we "**lose probability mass**"
  - See this visually!

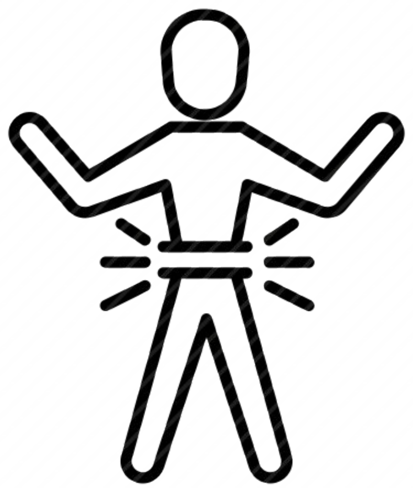


$$\underbrace{0.5 \times 1 \times 1 \times 1 \times 1 \times \ldots}_{\text{infinitely many}} = 0.5$$

ETH zürich

1 Tight and Non-Tight Language Models

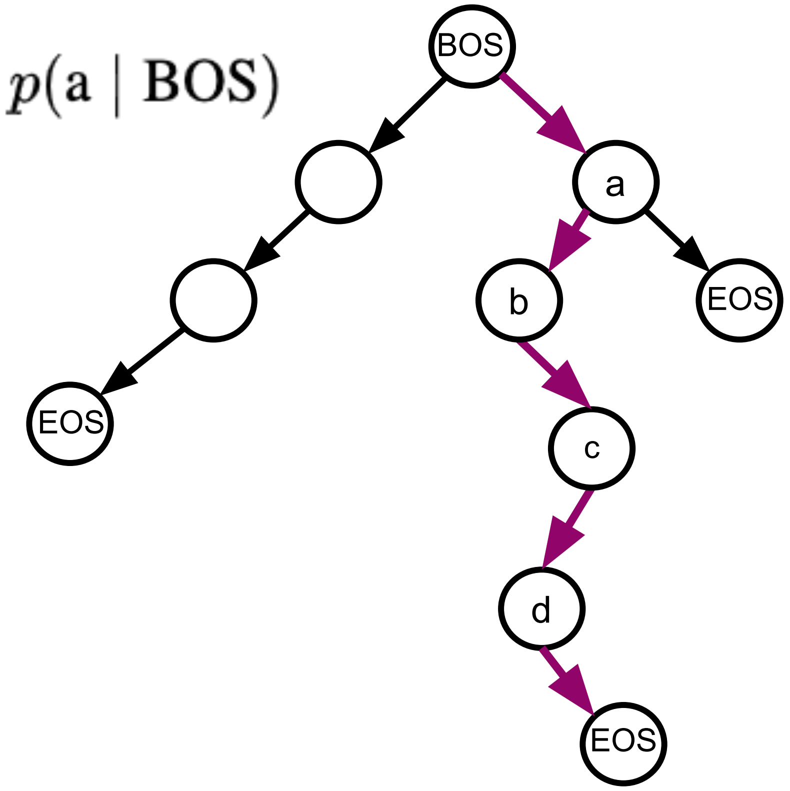
- A locally normalized language model that sums to 1 is called **tight**
- A **non-tight** language model is looses probability to infinitely long structures
- Generally, we ensure a model is tight by forcing $p(\text{EOS} \mid \text{parent}) > 0$ for every parent node.



1 Applying the Chain Rule of Probability to a Prefix Tree

- By the chain rule of probability, the joint factorizes into conditional probabilities

$$p(\mathrm{BOS\,a\,b\,c\,d\,EOS}) = p(\mathrm{EOS} \mid \mathrm{d})p(\mathrm{d} \mid \mathrm{cba})\,p(\mathrm{c} \mid \mathrm{ba})\,p(\mathrm{b} \mid \mathrm{a})\,p(\mathrm{a} \mid \mathrm{BOS})$$



- **Big Question**: How do we estimate the locally normalized conditional probability distributions *p*'s?

ETH zürich

1 Conditional Language Modeling

- We may not only be interested in modelling a languag ere **y** ∈ V*
- Instead, we may condition **y** on an **x**

Covered Later in the Course!!!

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{\text{e}(\mathbf{y},\mathbf{x})}{\sum_{\mathbf{y}' \in V^*} \exp \text{score}(\mathbf{y}',\mathbf{x})}$$

- In **machine trans on**, **x** is source-language text, and **y** is the target-language text
- In **speech recognition**, **x** is an audio signal, and **y** is target-language text
- In **summarization**, **x** is a longer text, and **y** is a shorter text
- In **dialogue systems**, **x** is a user's input and **y** is the agent's response

ETH zürich

n-gram Language Modeling

2 What is the *n*-gram assumption?

- How do we estimate *p*?

1. ***n*-gram assumption**

$$p(y_t \mid \mathbf{y}_{<t}) \overset{\text{def}}{=} p(y_t \mid \underbrace{y_{t-1}, \ldots, y_{t-n+1}}_{\text{call this the history } h})$$

Key Idea: We enforce a ***finite number of histories*** to make the modeling easier

ETH zürich

2

What is the *n*-gram assumption?

- How do we estimate *p*?

2. Joint estimation of large number of log-linear models

$$p(y_t \mid \mathbf{y}_{<t}) = p(y_t \mid y_{t-1}, \ldots, y_{t-n+1}) = \mathrm{softmax}(h_{y_t})$$

Key Idea: Tie the parameters between contexts.

2 *n*-gram Language Models

***n*-gram Language Model**

- We would need to model $|V|^{T-1}$ contexts. Such a distribution cannot be estimated from any realistic sample of text.
- To solve this problem, *n*-gram models make a crucial simplifying approximation: they condition on only the past $n-1$ words.

$$p(y_t \mid y_{t-1}, \ldots,$$

We need appropriate BOS padding here

For whole sentences:

$$p(y_1, \ldots, y_T) \overset{\text{def}}{=} \prod_{t=1}^{T} p(y_t \mid y_{t-1}, \ldots, y_{t-n+1})$$

2 Bengio et al. (2003): A Neural Probabilistic Language Model

- Bengio et al. (2003) proposed to model the conditional language model as a neural network

$$p(y_t \mid y_{t-1}, \ldots, y_1) \overset{\text{def}}{=} p(y_t \mid y_{t-1}, \ldots, y_{t-n+1})$$

- Was their model the first attempt to do neural language modeling?
  - No! But, they got it to work :-)

- But 2003 was so long okay, didn't the deep learning wave hit NLP around 2015?
  - Yes, it did. The neural models gave state-of-the-art performance in 2003
  - But, they were too slow: the softmax over the vocabulary slowed it down

2

Bengio et al. (2003): A Neural Probabilistic Language Model

- Journal version published in JMLR: https://jmlr.org/papers/volume3/tmp/bengio03a.pdf

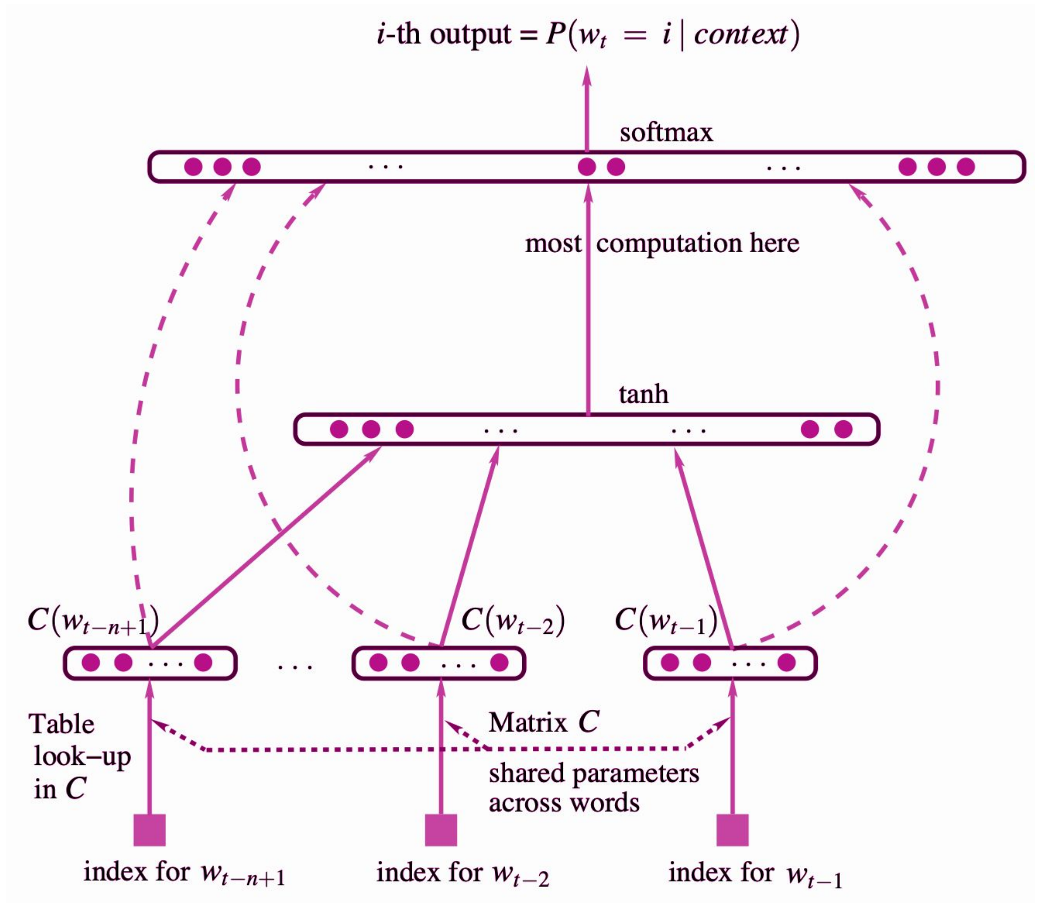
Problem: Curse of dimensionality

- **discrete random variables:** model the joint distribution between many discrete random variables (e.g. words in a sentence)
  - to model the joint distribution of 10 consecutive words with a vocabulary *V* of size 100,000 there are potentially $100{,}000^{10} - 1 = 10^{50} - 1$ free parameters

- **continuous variables:** however, if we use a MLP, we can reduce the number of parameters and increase the ability of the model to generalize

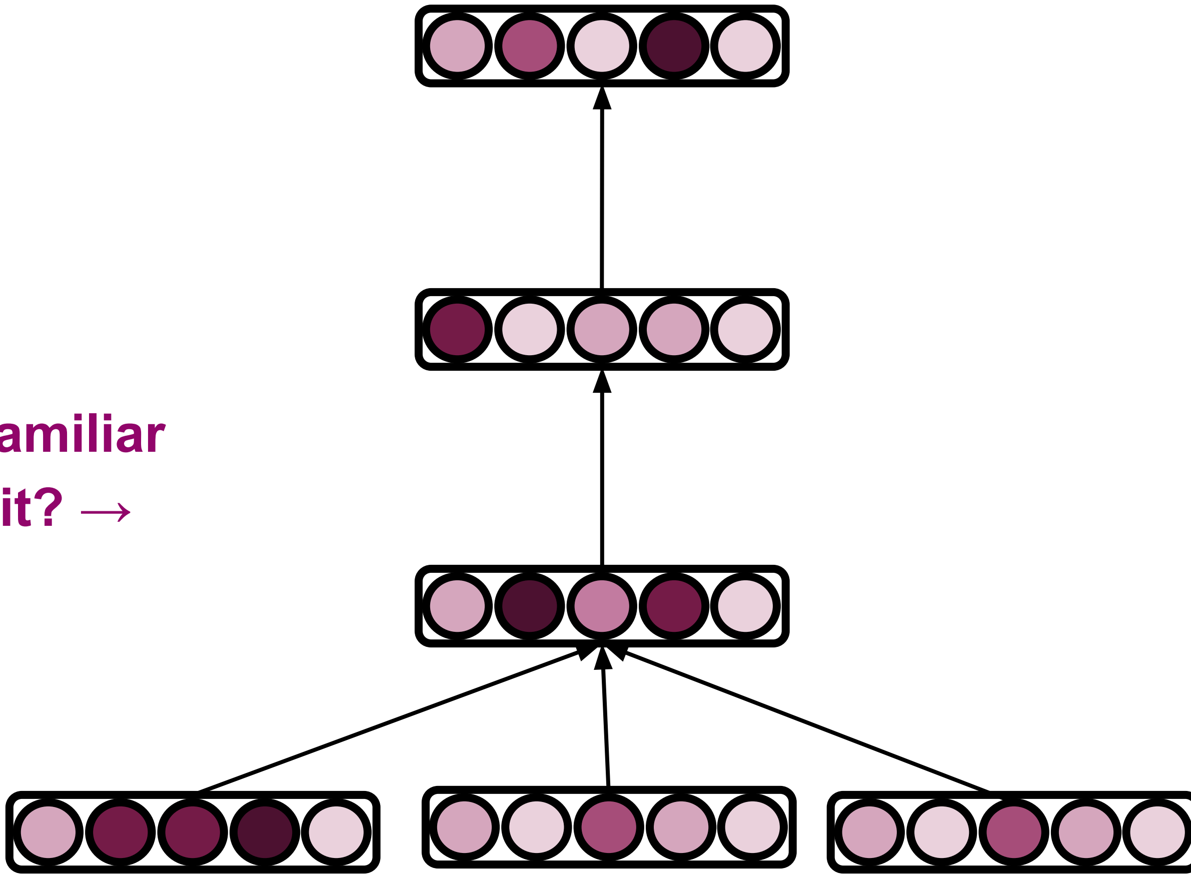
② Bengio et al. (2003): A Neural Probabilistic Language Model

Key idea: Use word embeddings in a MLP



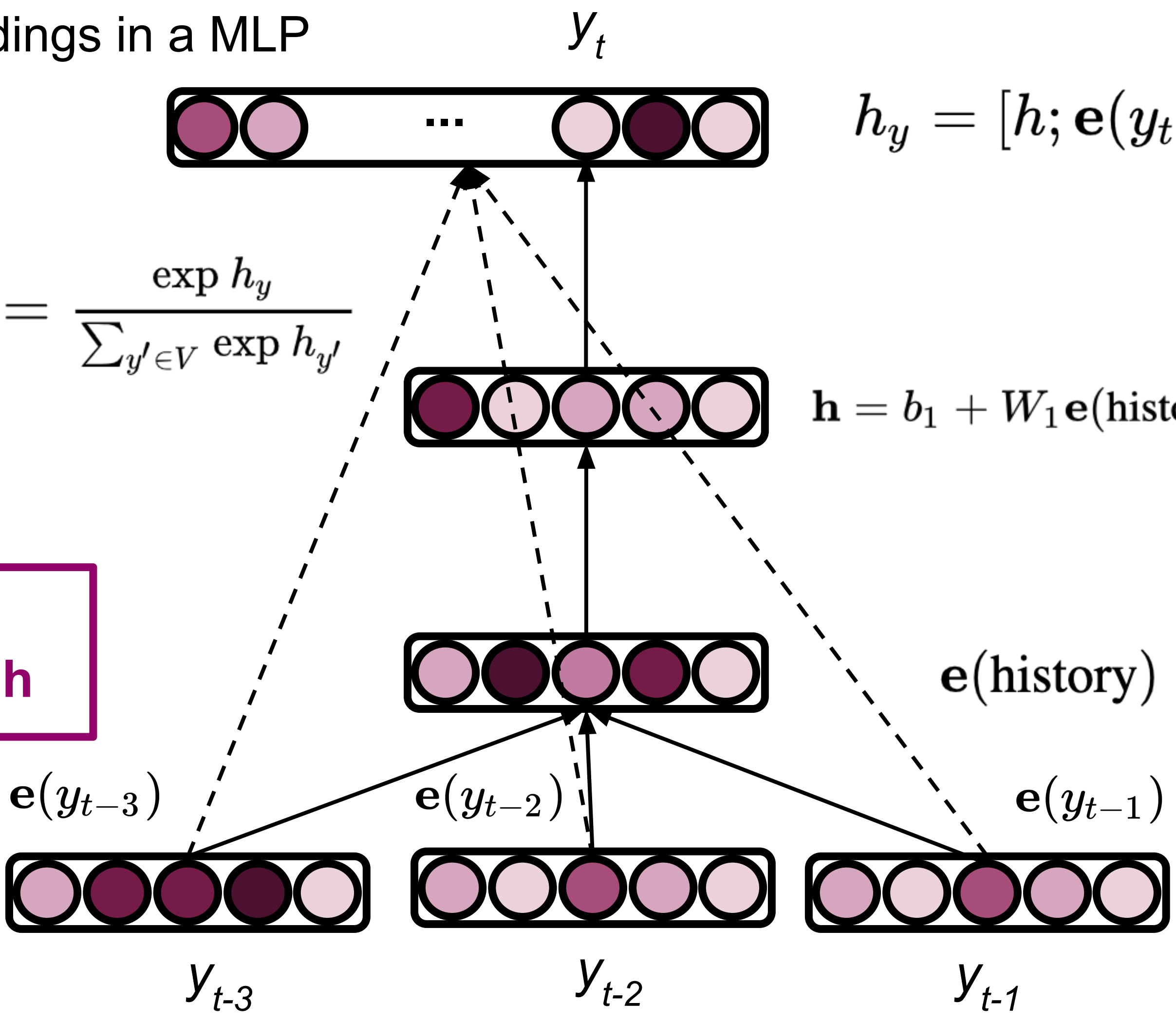
https://jmlr.org/papers/volume3/tmp/bengio03a.pdf

seems familiar doesn't it? →



② Bengio et al. (2003): A Neural Probabilistic Language Model

Key idea: 3 word embeddings in a MLP



$$h_y = [h; \mathbf{e}(y_{t-3}); \mathbf{e}(y_{t-2}); \mathbf{e}(y_{t-1})]$$

$$p(y_t \mid y_{t-1}, y_{t-2}, y_{t-3}) = \frac{\exp h_y}{\sum_{y' \in V} \exp h_{y'}}$$

$$\mathbf{h} = b_1 + W_1 \mathbf{e}(\text{history}) + W_2 \tanh(b_2 + W_3 \mathbf{e}(\text{history}))$$

$$\mathbf{e}(\text{history}) = [\mathbf{e}(y_{t-1}); \mathbf{e}(y_{t-2}); \mathbf{e}(y_{t-3})]$$

Recall the subscript *y* indexes into the vector **h**

Vector Concatenation

ETH zürich

2 Bengio et al. (2003): A Neural Probabilistic Language Model

Idea: Fighting the Curse of Dimensionality with Distributed Representations

1. Associate with each word in the vocabulary a distributed *word feature vector* (a real-valued vector in R^m),
2. Express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence
3. Learn simultaneously the *word feature vectors* and the parameters of that *probability function*.

Idea 1 seems familiar doesn't it? → $\mathbf{e}(\text{history}) = [\mathbf{e}(y_{t-1}); \mathbf{e}(y_{t-2}); \mathbf{e}(y_{t-3})]$

Idea 2 seems familiar doesn't it? → $\mathbf{h} = b_1 + W_1 \mathbf{e}(\text{history}) + W_2 \tanh(b_2 + W_3 \mathbf{e}(\text{history}))$

Idea 3 seems familiar doesn't it? → $p(y_t \mid y_{t-1}, y_{t-2}, y_{t-3}) = \text{softmax}\big(\boldsymbol{\theta} \cdot \mathbf{f}(e(\text{history}))\big)_{y_t}$

https://jmlr.org/papers/volume3/tmp/bengio03a.pdf

2

Bengio et al. (2003): A Neural Probabilistic Language Model

“What went well?”

- number of free parameters **scales linearly** with V
- **only scales linearly** with the order n:

“Even better if!”

- the scaling factor could be reduced to sub-linear if more sharing structure were introduced, e.g. using a **recurrent neural network**

https://jmlr.org/papers/volume3/tmp/bengio03a.pdf

2 Bengio et al. (2003): A Neural Probabilistic Language Model

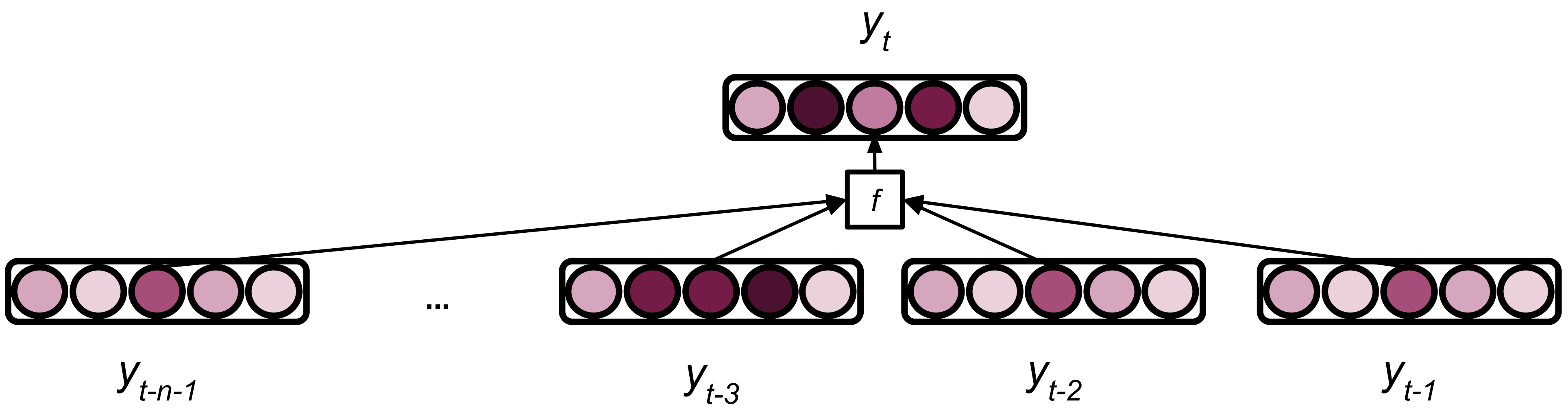
- We estimate the parameters by maximizing the log-likelihood

- We compute the gradient through backpropagation

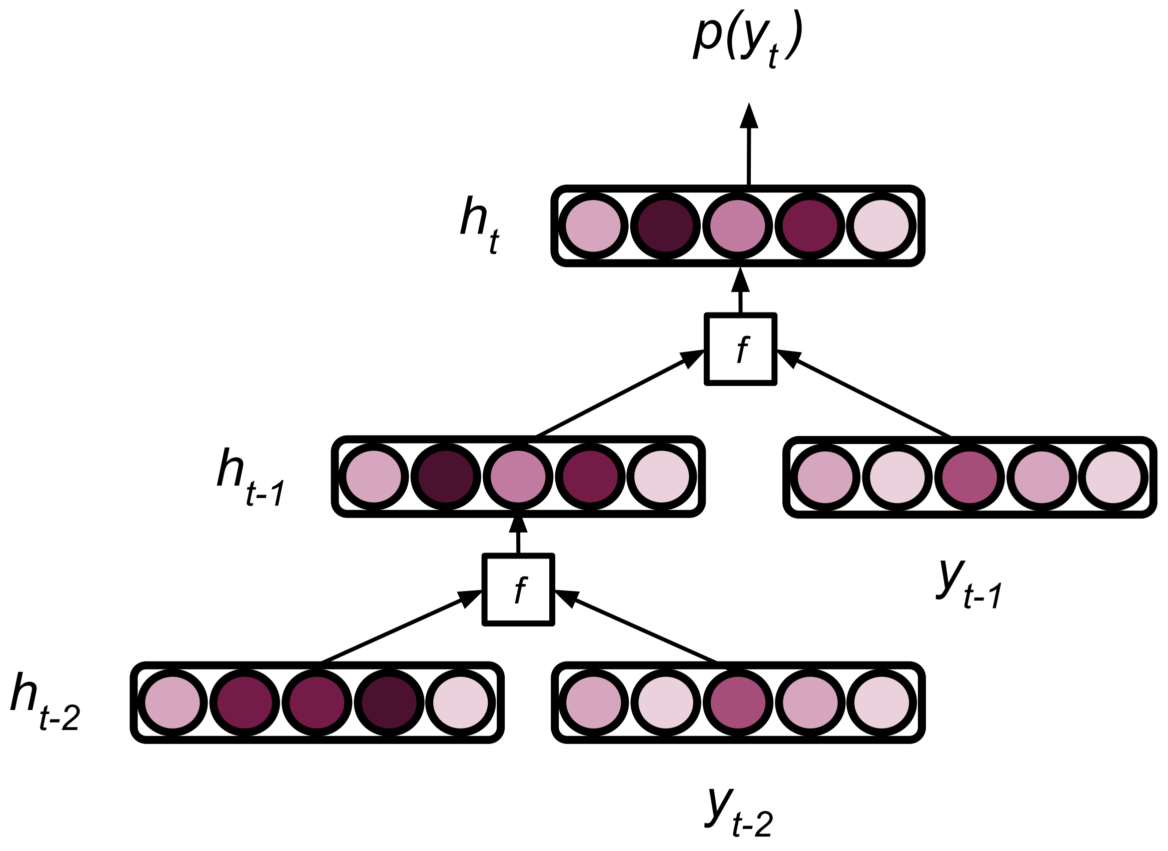
Recurrent Neural Networks

3 Language Modelling using Recurrent Neural Networks

- Bengio et al. (2003) give a neural parameterization of an *n*-gram model

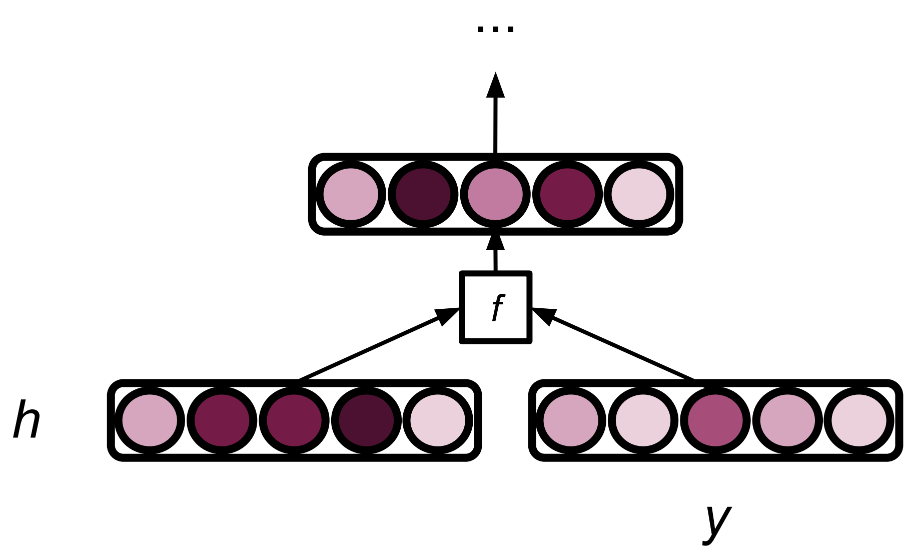


3 Language Modeling using Recurrent Neural Networks



- Instead of a fixed context considered at every step, imagine we have two inputs to the model at every timestep
    - Distributed representation for current token *(y)*
    - Recurrent connection (*h*)
- The recurrent connection is the distributed representation of the past token plus the past value of the recurrent connection...
- This is a language model with **infinite context**! (in theory)

3 Language Modelling using Recurrent Neural Networks

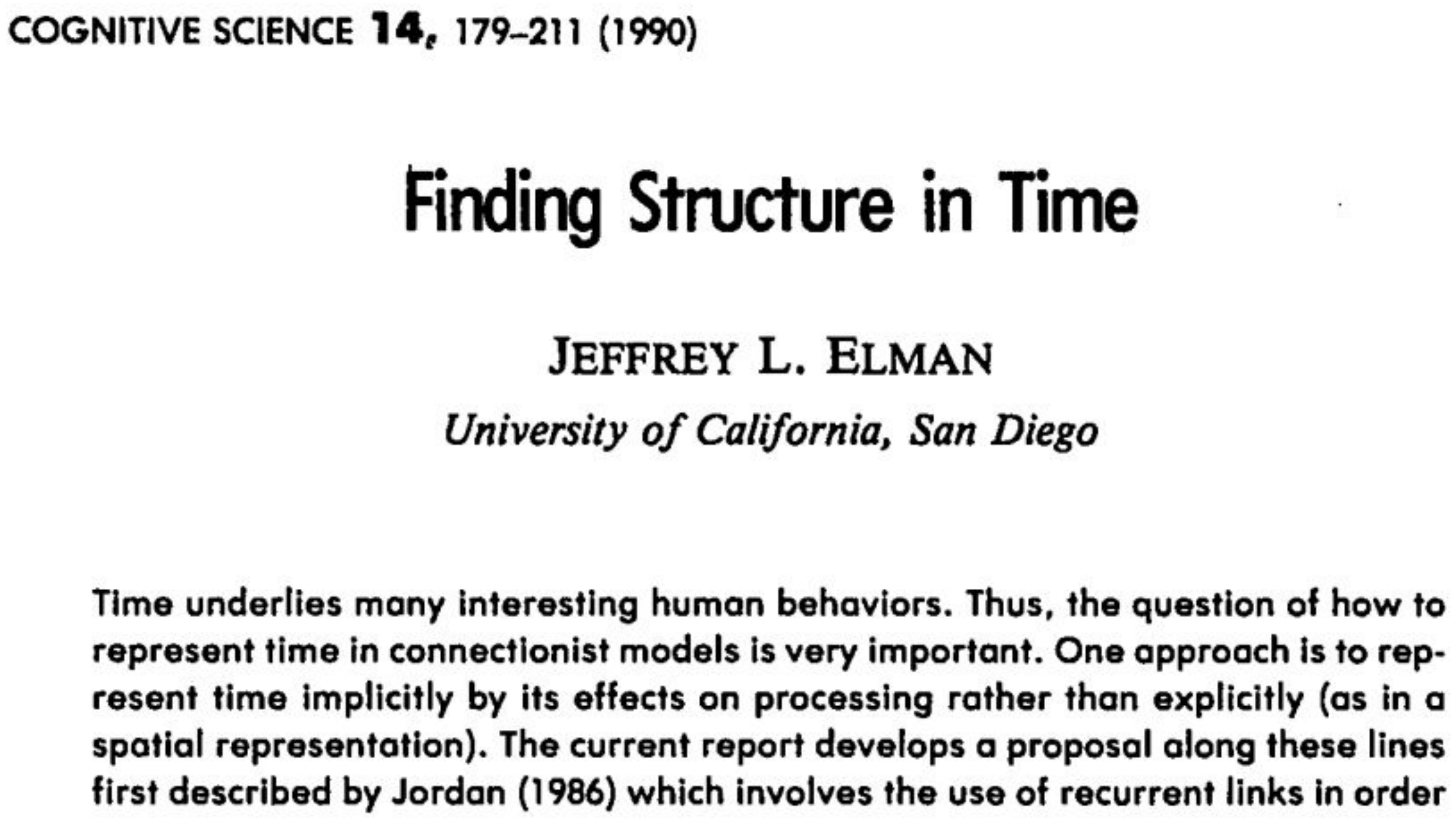


- The function *f* describes how we combine the recurrent part of the model with the distributed representation of the input at the current time step
- These are referred to as **cells**. Three that we will cover are:
    - RNN
    - GRU
    - LSTM
- There are <u>many</u> variants of these

③ Language Modelling using Recurrent Neural Networks

Elman Networks (Vanilla RNNs)

- Historical note: RNNs have been around for a while!



COGNITIVE SCIENCE **14**, 179–211 (1990)

Finding Structure in Time

JEFFREY L. ELMAN

University of California, San Diego

Time underlies many interesting human behaviors. Thus, the question of how to represent time in connectionist models is very important. One approach is to represent time implicitly by its effects on processing rather than explicitly (as in a spatial representation). The current report develops a proposal along these lines first described by Jordan (1986) which involves the use of recurrent links in order

3 Recurrent Neural Networks

Key ideas:

- incorporate arbitrarily distant contextual information
- treat word prediction as a discriminative learning task

- We reparameterize the distribution p(y_t | $\mathbf{y}_{<t}$) as a function of two dense *K*-dimensional numerical vectors

$$p(y_t \mid \mathbf{y}_{<t}) = p(y_t \mid y_1, \ldots, y_{t-1}) = \frac{\exp(\boldsymbol{\theta}_{y_t} \cdot \mathbf{h}_t)}{\sum_{y' \in V} \exp(\boldsymbol{\theta}_{y'} \cdot \mathbf{h}_t)}$$

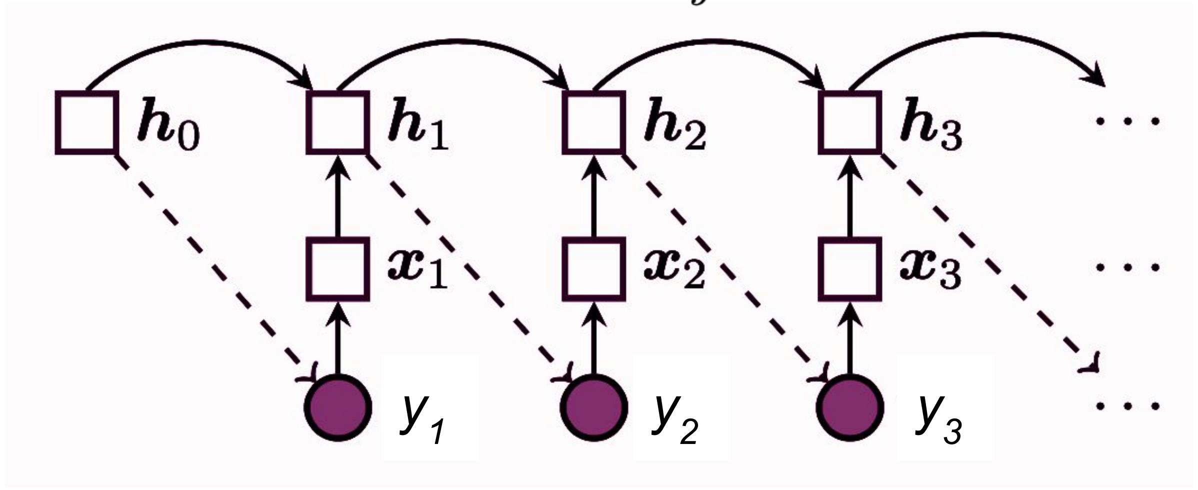
$\boldsymbol{\theta}_y \in \mathbb{R}^d$ **(output) word vectors** are parameters of the model

$\mathbf{h}_t \in \mathbb{R}^d$ **context vectors** computed in various ways, depending on the model, but crucially they incorporate **all previous context**

3 Recurrent Neural Networks

- In a general **Recurrent Neural Network (RNN; Mikolov et al., 2010)**, we recurrently update the context vectors $\mathbf{h}_t$ while moving through the sequence.
- Broadly speaking, RNN language models are defined as

$$\mathbf{x}_t = \boldsymbol{\phi}_{y_t}$$
$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$$
$$p(y_t \mid \mathbf{y}_{<t}) = \frac{\exp(\boldsymbol{\theta}_{y_t} \cdot \mathbf{h}_t)}{\sum_{y' \in V} \exp(\boldsymbol{\theta}_{y'} \cdot \mathbf{h}_t)}$$



where

$\left.\begin{array}{l}\Phi \text{ is a matrix of input embeddings} \\ \Theta \text{ is a matrix of output embeddings}\end{array}\right\}$ Sometimes these are shared (aka. **tied**)

x_t denotes the embedding for word y_t

f is the cell we want to use (e.g., RNN, LSTM)

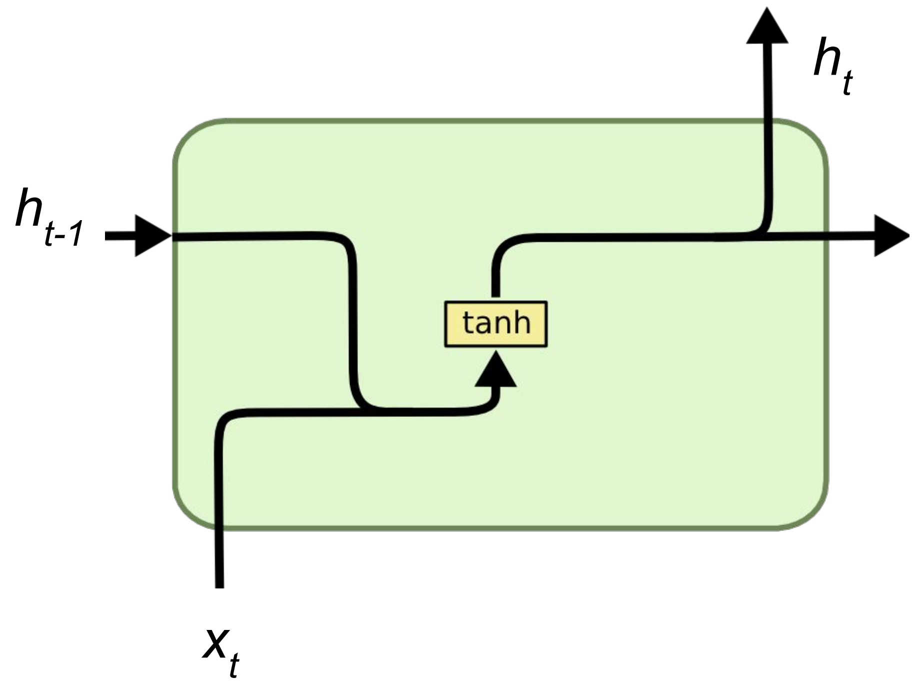
- The mapping from y_t to x_t via Φ is sometimes referred to as a **lookup layer**
- The reason is that this can be seen as a selecting a vector from a list/dictionary/table of word-vector pairs

ETH zürich

3 Recurrent Neural Networks

(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.



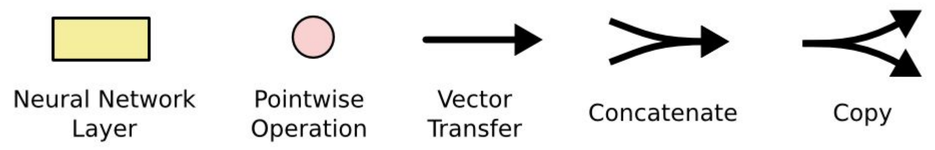
Elman recurrence (Elman, 1990):

$$\mathbf{h}_t = g(\Theta\,\mathbf{h}_{t-1} + \mathbf{x}_t)$$

Variant:

$$\mathbf{h}_t = g(\Theta[\mathbf{h}_{t-1}\,;\mathbf{x}_t])$$

where $\Theta \in \mathbb{R}^{d\times d}$ is a recurrence matrix,
g is a non-linearity (e.g., tanh)

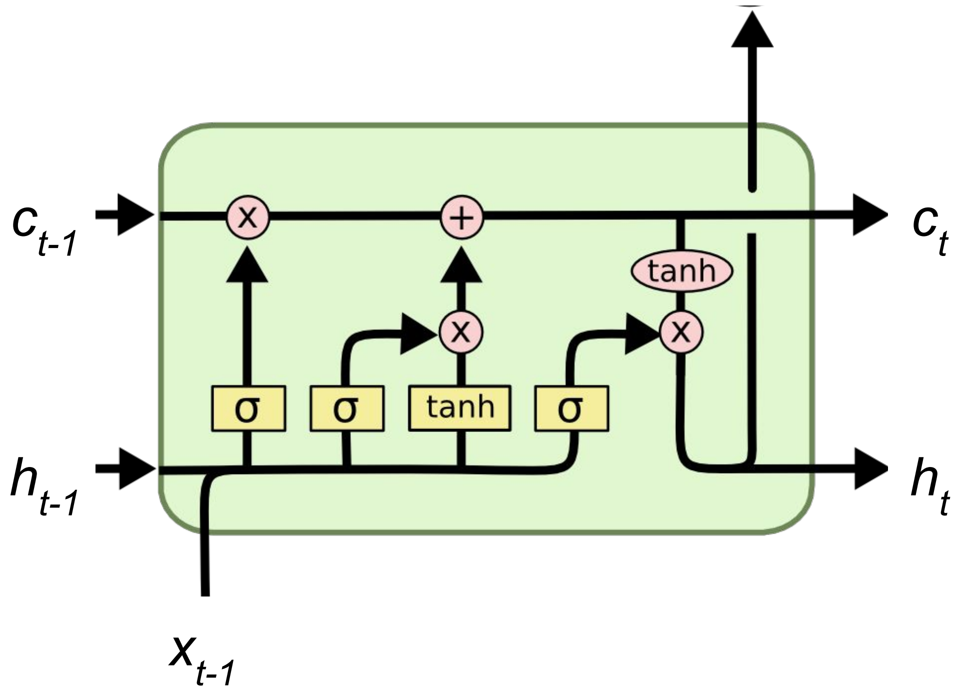


from Christopher Olah's blog, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

3 Recurrent Neural Networks

Long short-term memory (LSTM)

LSTMs (Hochreiter and Schmidhuber, 1997) have the form



$$\mathbf{f}_t = \sigma(\Theta_f[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_f)$$ **forget gate**

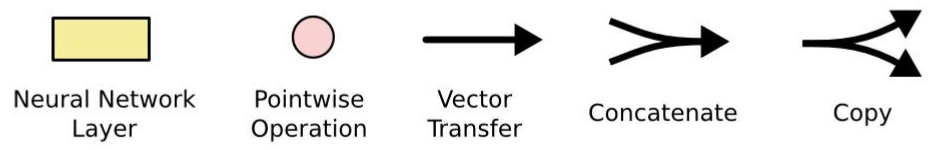
$$\mathbf{i}_t = \sigma(\Theta_i[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_i)$$ **input gate**

$$\tilde{\mathbf{c}}_t = \tanh(\Theta_c[\mathbf{h}_{t-1};\mathbf{x}_t] + \mathbf{b}_c)$$ **update candidate**

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t$$ **update cell state**

$$\mathbf{o}_t = \sigma(\Theta_o[\mathbf{h}_{t-1},\mathbf{x}_t] + \mathbf{b}_o)$$ **output gate**

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$ **update hidden state**

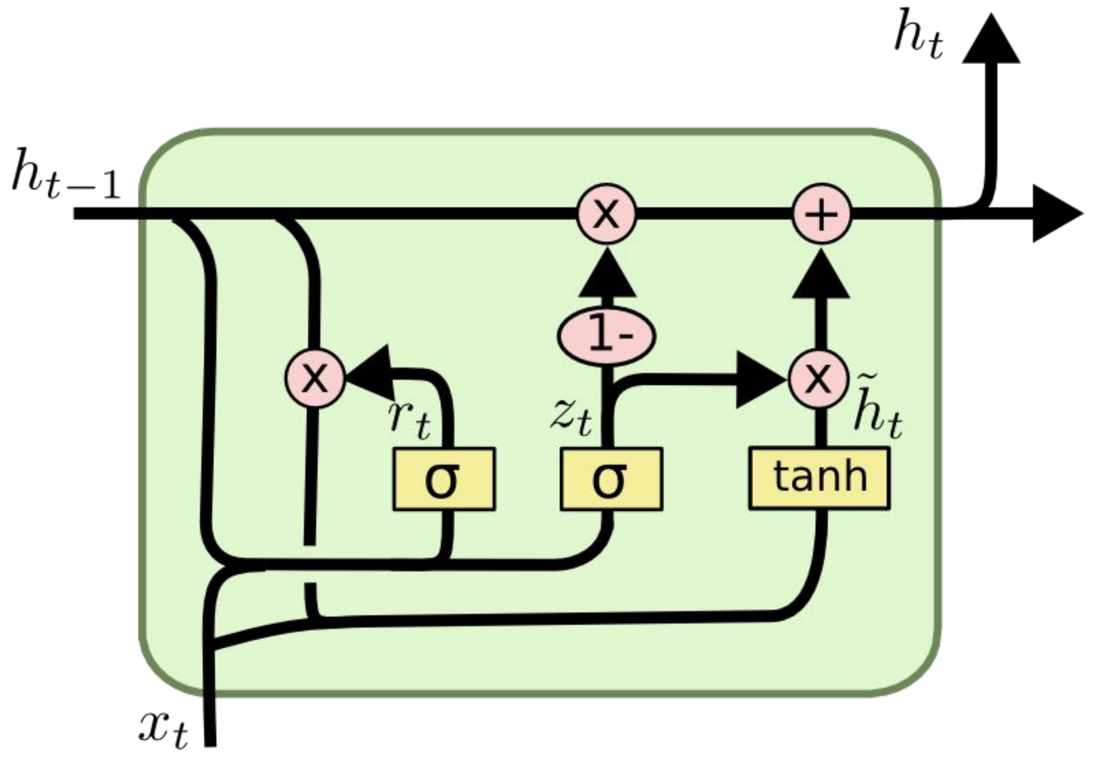


from Christopher Olah's blog, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

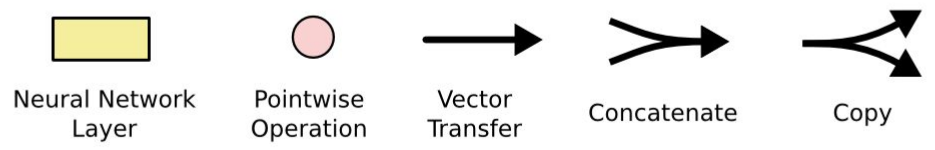
Recurrent Neural Networks

Gated Recurrent Neural Networks (GRUs)

GRUs (Cho et al., 2014) are another variant of RNN cell that tries to solve the vanishing/exploding gradient problem



$$
\begin{aligned}
\mathbf{z}_t &= \sigma(\Theta_z[\mathbf{h}_{t-1};\mathbf{x}_t]) \\
\mathbf{r}_t &= \sigma(\Theta_r[\mathbf{h}_{t-1};\mathbf{x}_t]) \\
\tilde{\mathbf{h}}_t &= \tanh(\Theta_{\tilde{h}}[\mathbf{r}_t \odot \mathbf{h}_{t-1};\mathbf{x}_t]) \\
\mathbf{h}_t &= (\mathbf{1} - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t
\end{aligned}
$$



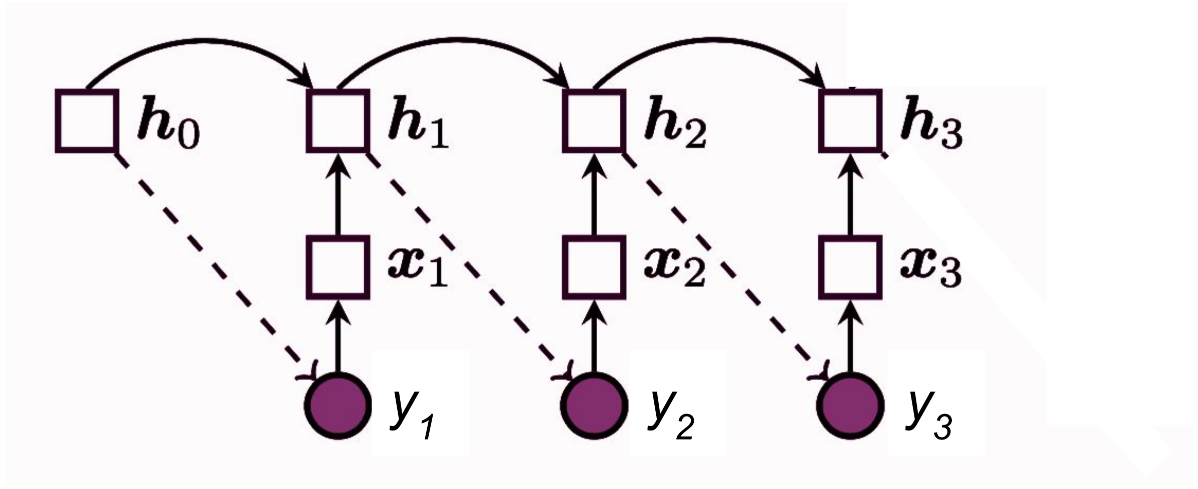
from Christopher Olah's blog, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

3 Backpropagation Through Time

- Recurrent neural networks are trained through backpropagation ***through time*** (Werbos 1974)
  - What is backpropagation ***through time***?
  - It’s the same as backpropagation, but given a fancier name

- The idea is that you first unfold the network and then you run the algorithm we learned in Lecture #2

- For any input sentence, the unrolled network is finite and can be backproped through



3 Backpropagation Through Time is Just Backpropagation

Suppose that our recurrent cell is defined as (aka. vanilla Elman RNN)

$$\mathbf{h}_t = g(\Theta \mathbf{h}_{t-1} + \mathbf{x}_t)$$

where *g* is some differentiable activation function (e.g., ReLU, tanh)

The derivative of the recurrent state with respect to the recurrent parameters are

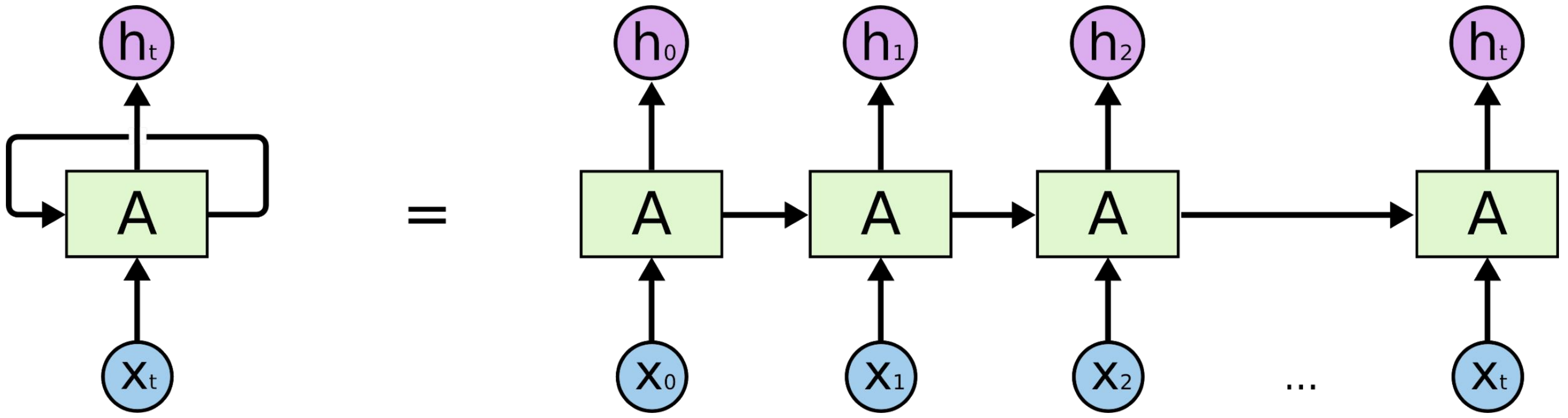
$$\frac{\partial h_{t,k}}{\partial \theta_{k,k'}} = g(x_{t,k} + \boldsymbol{\theta}_k \cdot \mathbf{h}_{t-1})(h_{t-1,k'} + \boldsymbol{\theta}_k \boxed{\cdot \frac{\partial \mathbf{h}_{t-1}}{\partial \theta_{k,k'}})} \longleftarrow$$

the recurrence manifests in the gradient

Hence we need to unroll our RNN over multiple timesteps, and backprop over the whole thing

from Eisenstein, Introduction to Natural Language Processing, Ch. 6

3 Backpropagation Through Time is Just Backpropagation



- Here *A* is whatever RNN cell we want to use (e.g., LSTM, RNN, etc.). Each timestep yields (i) an output and (ii) a recurrent connection.
- Backpropagating RNNs is the same as backpropagating through any neural network, except the parameters are shared across timesteps.
- This same idea can be used to, e.g., tie embeddings or reuse a filter in a CNN

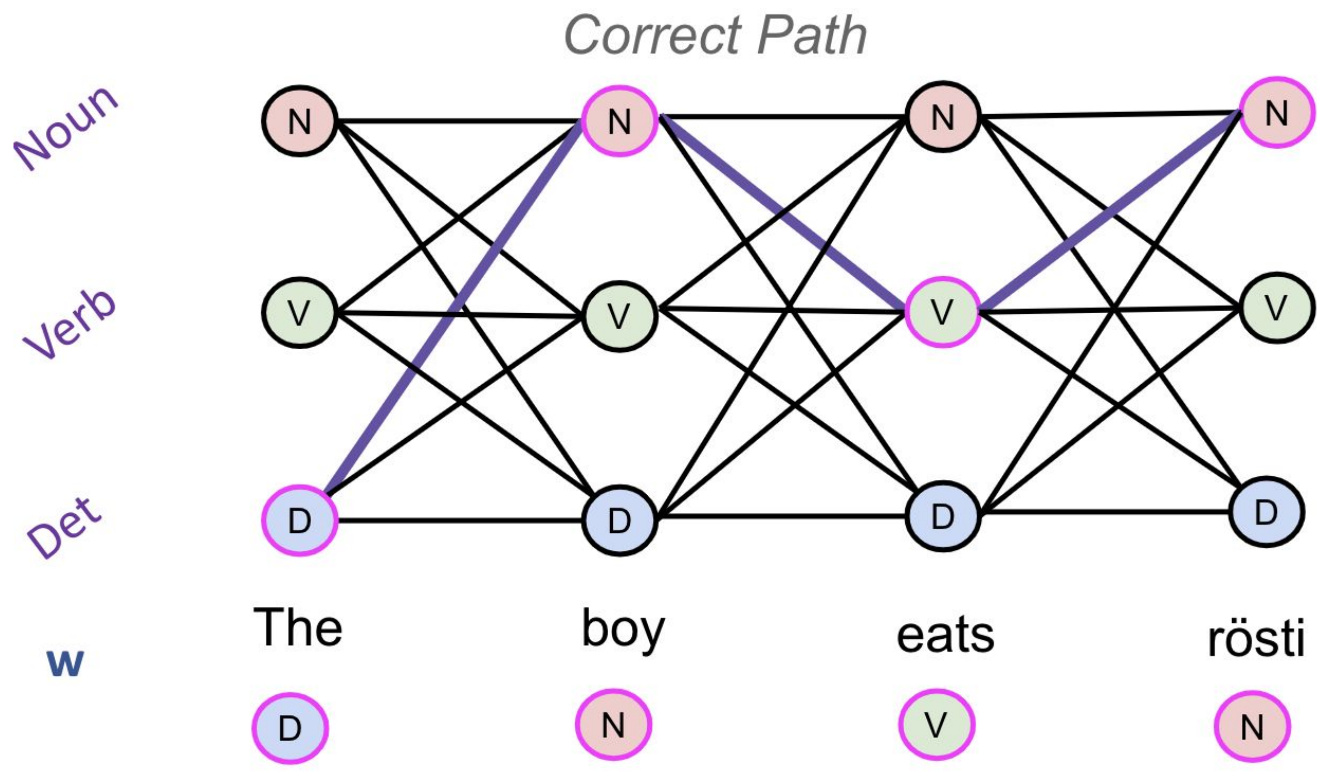
from Christopher Olah's blog, https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Sneak Preview of Next Lecture

Part-of-Speech Tagging with CRFs

More structured prediction!

- This time we show how to impose structural constraints on our model such that we can use a globally normalized scheme.
- Spoiler: Parts of this lecture may be reminiscent of our backpropagation lecture...



ETH zürich

Fin