School of CIT
Social Computing Research Group

Technical
University
of Munich

Natural Language Processing
IN2361

Prof. Dr. Georg Groh

Chapter 9
Deep Learning Architectures for Sequence Processing

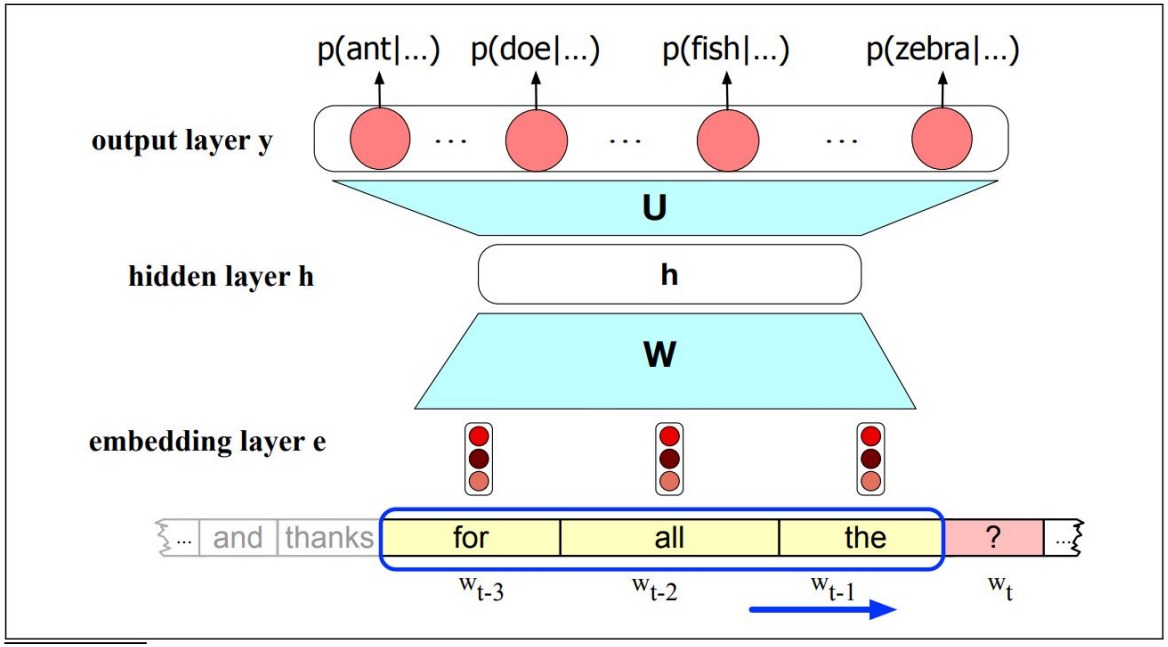
- content is based on [1] , [5]
- certain elements (e.g. equations or tables) were taken over or taken over in a modified form from [1] , [5]
- citations of [1] , [5] or from [1] , [5] are omitted for legibility
- errors are fully in the responsibility of Georg Groh
- BIG thanks to Dan and James for a great book!
- BIG thanks to Richard Socher, Chris Manning and their colleagues at Stanford for publishing materials [2],[5] of a great Deep NLP lecture

Processing Temporal Information

HMM has disadvantage integrating new features is combersa? IDK meaning

- Language is a temporal phenomenon.
- Sequence of spoken/written words
- Algorithms can reflect the temporal nature e.g. Viterbi-algorithm

- What about Feedforward Neural Networks?



Processing Temporal Information

- Feedforward neural networks are NOT temporal
  - Assumption of simultaneous access
  - Possible “fix”: sliding window approach
- Problems with fixed size input:
  - possibly limiting the contextual information

Never a fixed input size is enough. Also increasing it comes cost

- Two new solutions for the context problem:
  - Recurrent neural networks:
    - Representation of prior context
  - Transformer networks
    - Self-attention and positional encodings to relate words over distance

Fixed Window NN Language Models

~~as the proctor started the clock~~ the students opened their _____

discard

fixed window

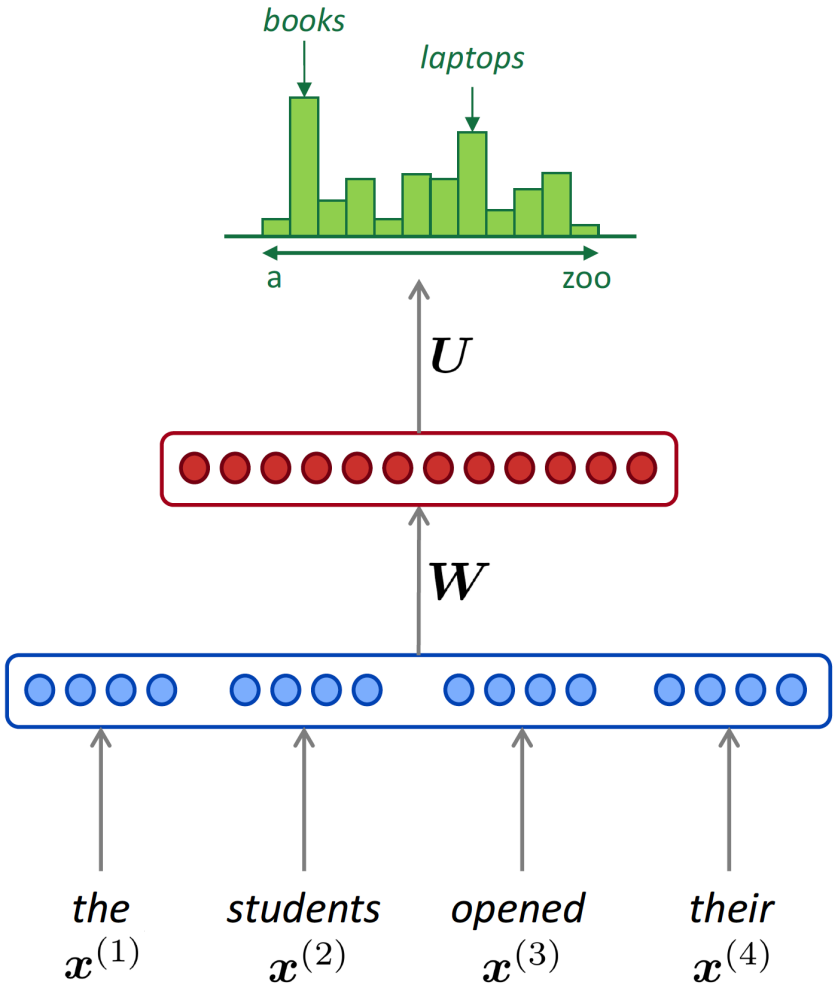
Improvements over *n*-gram LM:

- No sparsity problem
- Model size is O(*n*) not O(exp(*n*))

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges $\boldsymbol{W}$
- Window can never be large enough!
- Each $\boldsymbol{x}^{(i)}$ uses different rows of $\boldsymbol{W}$. We don't share weights across the window.

We need a neural architecture that can process *any length input*



Probabilistic Language Models Revisited

- Recall:
  - Language models predict the next word given a previous context

$$P(fish|Thanks\ for\ all\ the)$$

  - able to assign probabilities to sequences of words

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i|w_{<i})$$

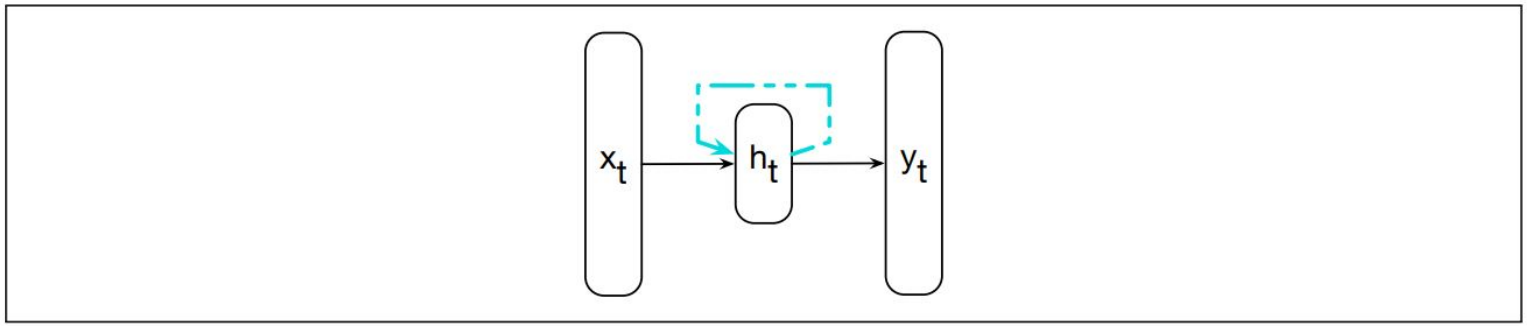
  - good language model: Predicts unseen text well
  - or: Good models are less surprised when encountering new words
  - how to measure: Perplexity of model θ

$$\mathrm{PP}_\theta(w_{1:n}) = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P_\theta(w_i|w_{1:i-1})}}$$

Recurrent Neural Networks

- Definition:
  - Any network that contains a cycle within the network connection.
- result: value of some unit within the network is directly (or indirectly) dependent (as an additional input) on its own earlier outputs.

- early example: Elman Network (==basic RNN)

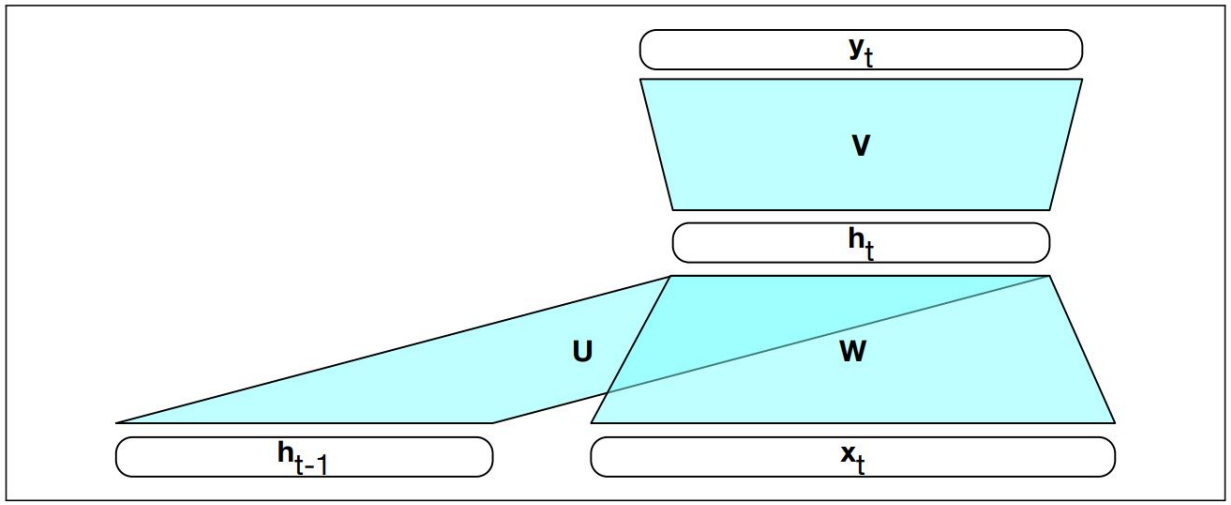


  - hidden layer contains a recurrent connection
  - sequences are processed one item at a time

Recurrent Neural Networks

- Hidden layer provides memory from the previous time step
- No fixed-length limit on the prior context

- Seemingly more complex, but:
  - Still operates on standard feedforward calculation
  - New weights **U** connecting to the previous hidden state



Inference in RNNs

- Forward inference via computation graph in principal same as FF Neural networks

$$\mathbf{h}_t = g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t)$$
$$\mathbf{y}_t = f(\mathbf{V}\mathbf{h}_t)$$

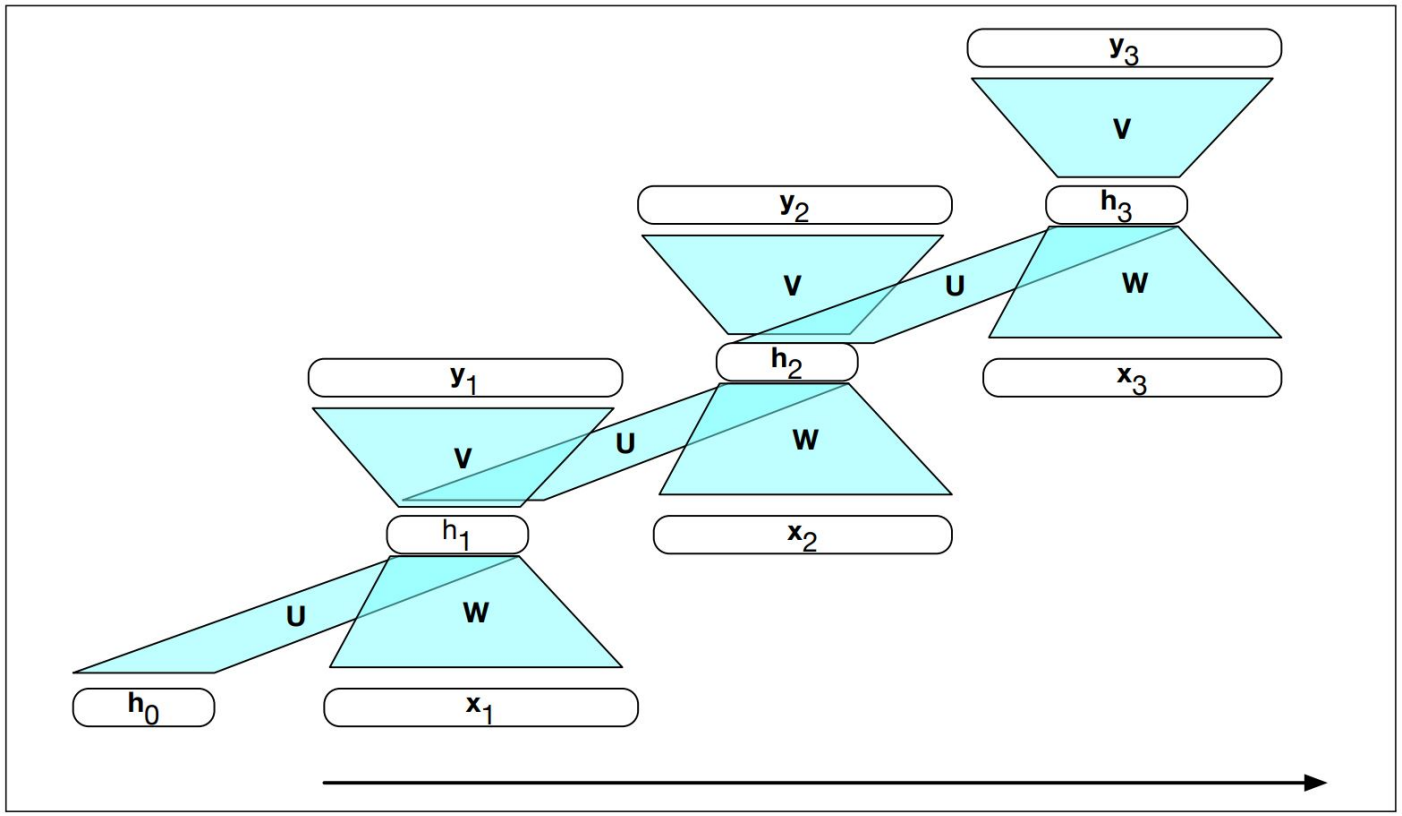
$$W \in \mathbb{R}^{d_h \times d_{in}}$$
$$U \in \mathbb{R}^{d_h \times d_h}$$
$$V \in \mathbb{R}^{d_{out} \times d_h}$$

- Incremental inference algorithm
- Weight matrices are shared during inference time

function FORWARDRNN($\mathbf{x}$, *network*) **returns** output sequence $\mathbf{y}$

$\mathbf{h}^0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH($\mathbf{x}$) **do**
  $\mathbf{h}_i \leftarrow g(\mathbf{U}\mathbf{h}_{i-1} + \mathbf{W}\mathbf{x}_i)$
  $\mathbf{y}_i \leftarrow f(\mathbf{V}\mathbf{h}_i)$
return y

Training RNNs



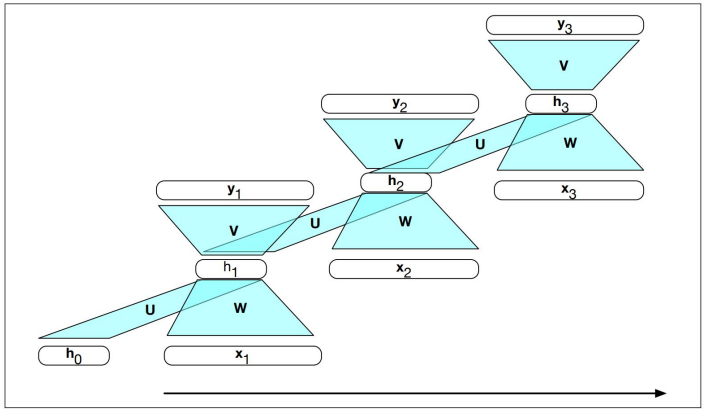
- Training using backpropagation to adjust weights **W**, **U** and **V**.
- However:
  - Assessing the error h_t requires computing its influence on current and future outputs.

teknik olarak istedigimiz windowu verebiliyoruz gibi gozukse de
10 15 kelimeden sonra eskileri unutmayi baslayacak

Training RNNs

- Solution Backpropagation Through Time:
  - Two-pass algorithm for training
  - First pass – forward inference computing h_t, y_t
  - Saving value of the hidden layer at each timestep
  - Second pass – reverse processing computing gradients

- Alternative: Unrolling into feedforward computational graph
  - but: More computationally expensive as input length increases
  - → limit unrolling length / depth



RNNs as Language Models

- RNN language modelling (Mikolov et al., 2010):
  - Predict the next word from current word and previous hidden state.

- Advantages:
  - No limited context
  - Easy forward inference

- Making use of a word embedding matrix **E** to encode the current word

$$
\begin{aligned}
\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\
\mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\
\mathbf{y}_t &= \text{softmax}(\mathbf{V}\mathbf{h}_t)
\end{aligned}
$$

RNNs as Language Models

$$\begin{aligned}\mathbf{e}_t &= \mathbf{E}\mathbf{x}_t \\ \mathbf{h}_t &= g(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{e}_t) \\ \mathbf{y}_t &= \text{softmax}(\mathbf{V}\mathbf{h}_t)\end{aligned}$$

- Vector $\mathbf{V}\mathbf{h}_t$ is a set of scores over the vocabulary given the evidence in the hidden state h.

- Probability of a word being the next word in the sequence

$$P(w_{t+1} = i | w_1, \ldots, w_t) = \mathbf{y}_t[i]$$

- Probability of the entire sequence:

$$\begin{aligned}P(w_{1:n}) &= \prod_{i=1}^{n} P(w_i | w_{1:i-1}) \\ &= \prod_{i=1}^{n} \mathbf{y}_i[w_i]\end{aligned}$$

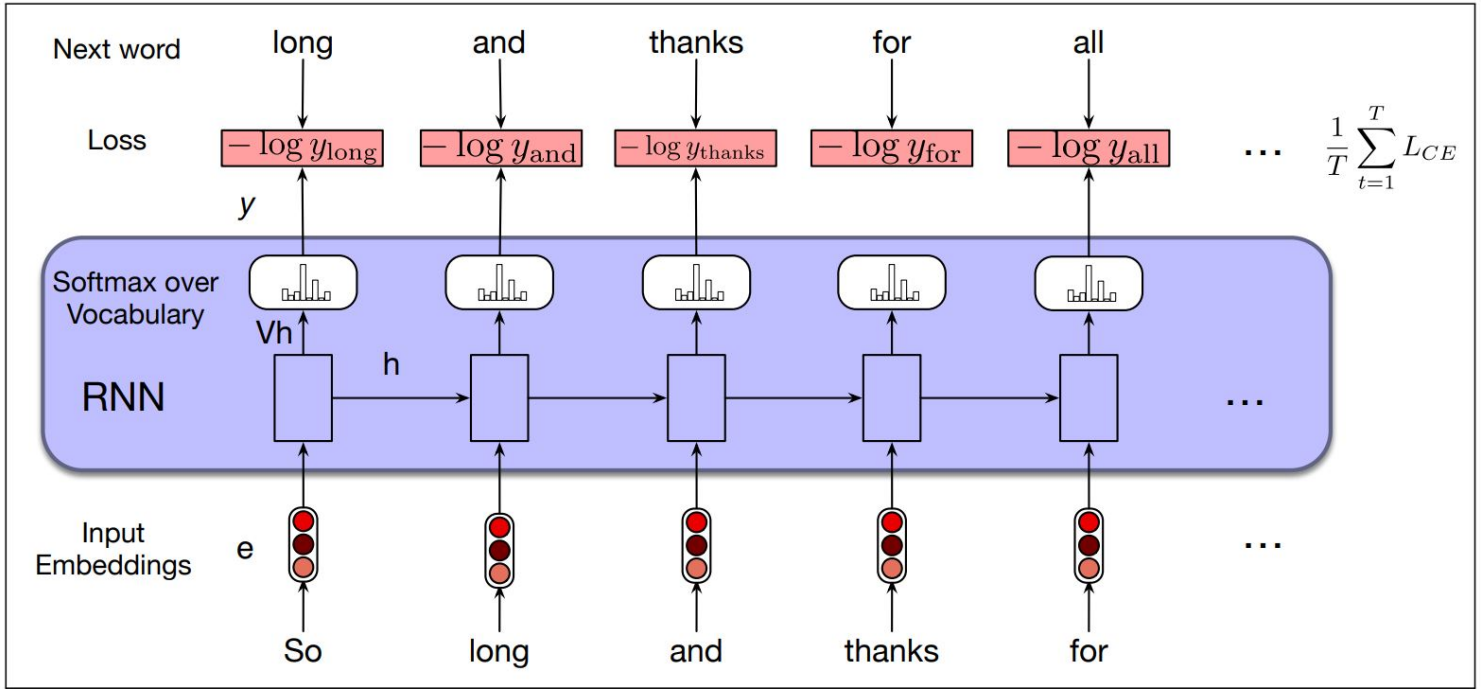
where $y_i[w_i]$ is the probability of the true word w_i at time i

RNNs as Language Models

- Training using a corpus of text by predicting the next word
- Minimizing cross-entropy loss $L_{CE} = -\sum_{w \in V} \mathbf{y}_t[w] \log \hat{\mathbf{y}}_t[w]$

$$L_{CE}(\hat{\mathbf{y}}_t, \mathbf{y}_t) = -\log \hat{\mathbf{y}}_t[w_{t+1}]$$

- problem: models make mistakes predicting word after word and make up improbably sequences.
- solution: Input is always the correct sequence (teacher forcing)



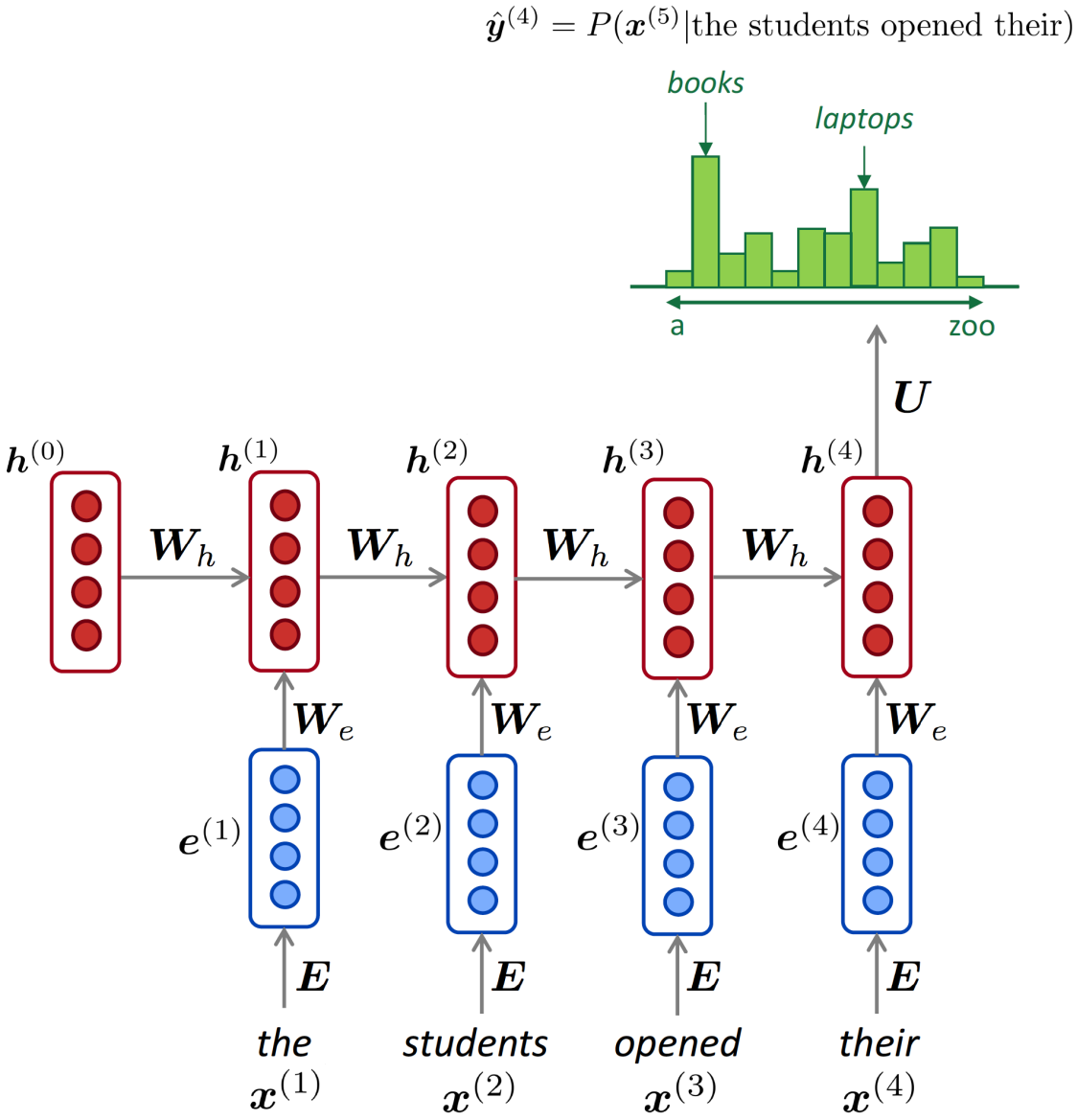
RNN Language Model

RNN **Advantages**:

- Can process any length input
- Model size doesn't increase for longer input
- Computation for step *t* can (in theory) use information from many steps back
- Weights are shared across timesteps → representations are shared

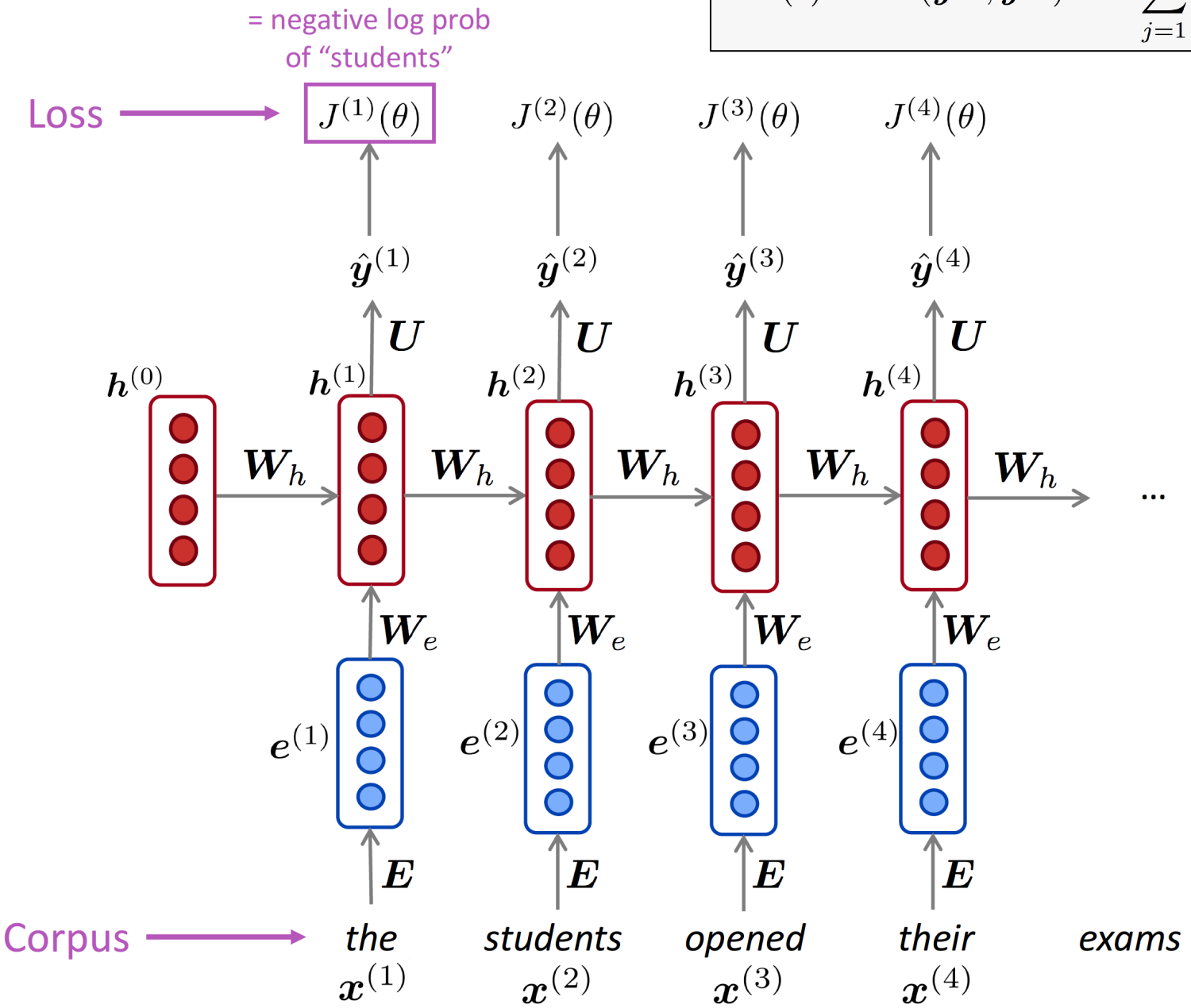
RNN **Disadvantages**:

- Recurrent computation is slow not paralel
- In practice, difficult to access information from many steps back

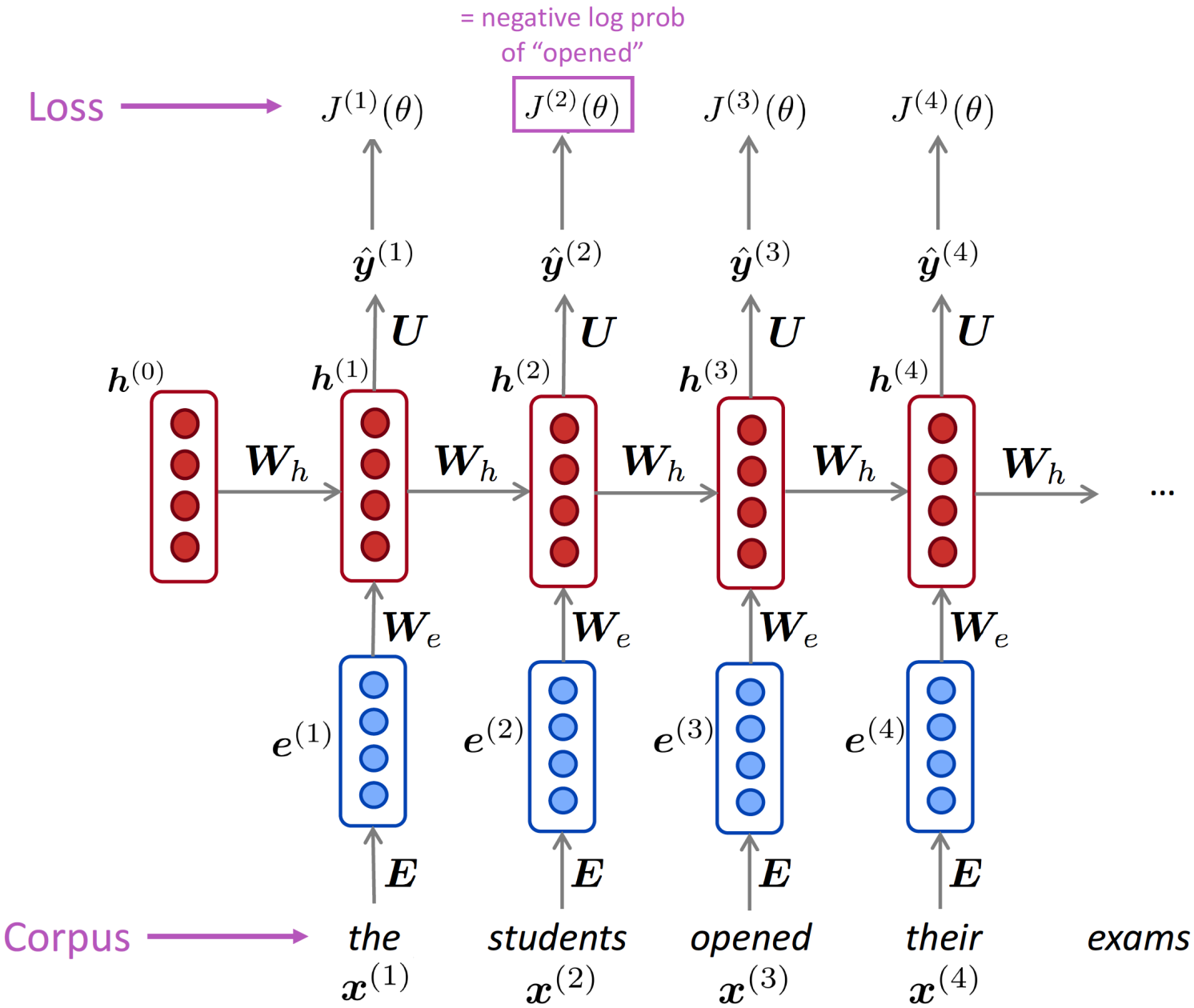


RNN LM: Training

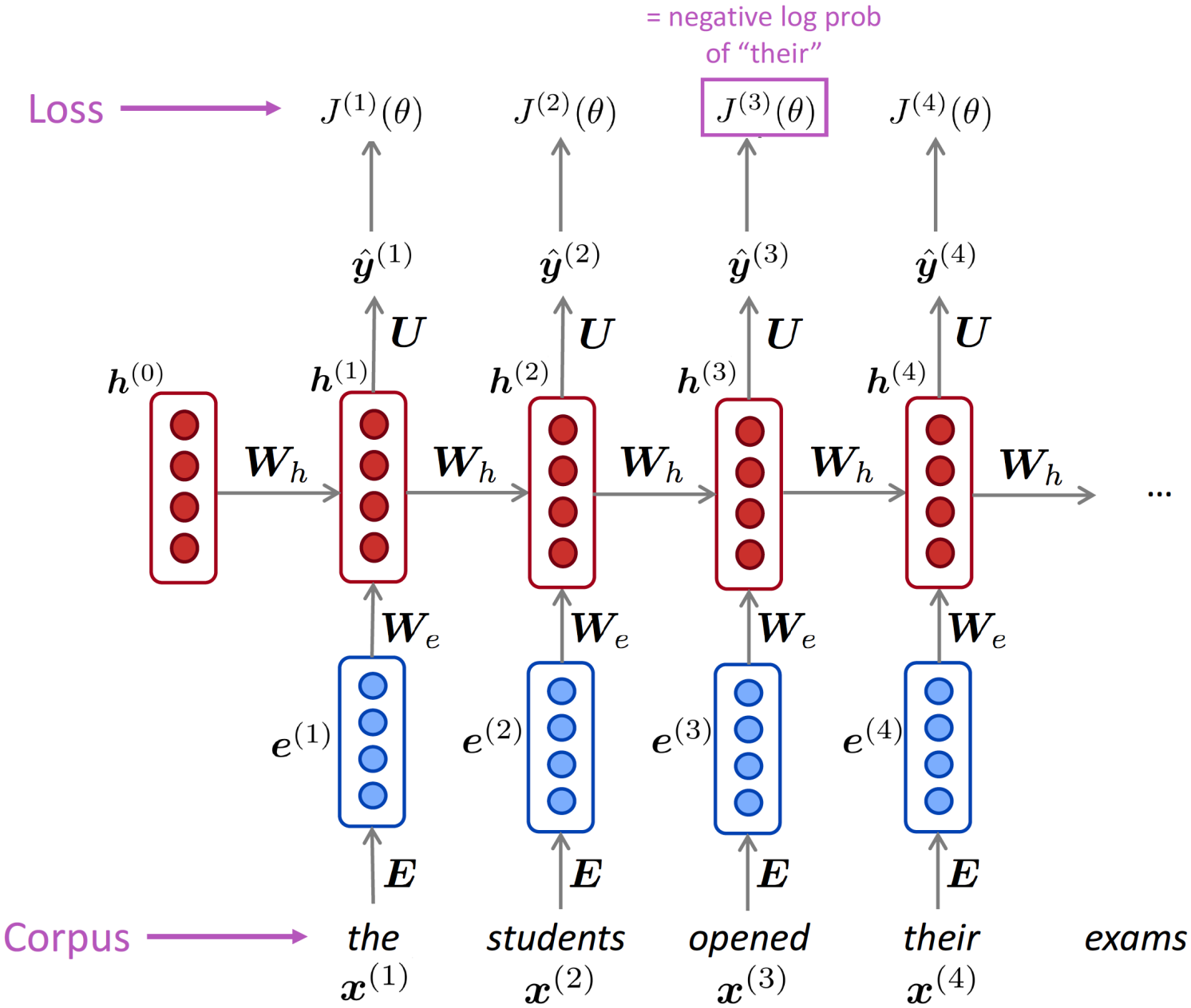
$$J^{(t)}(\theta) = CE(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}) = -\sum_{j=1}^{|V|} y_j^{(t)} \log \hat{y}_j^{(t)}$$



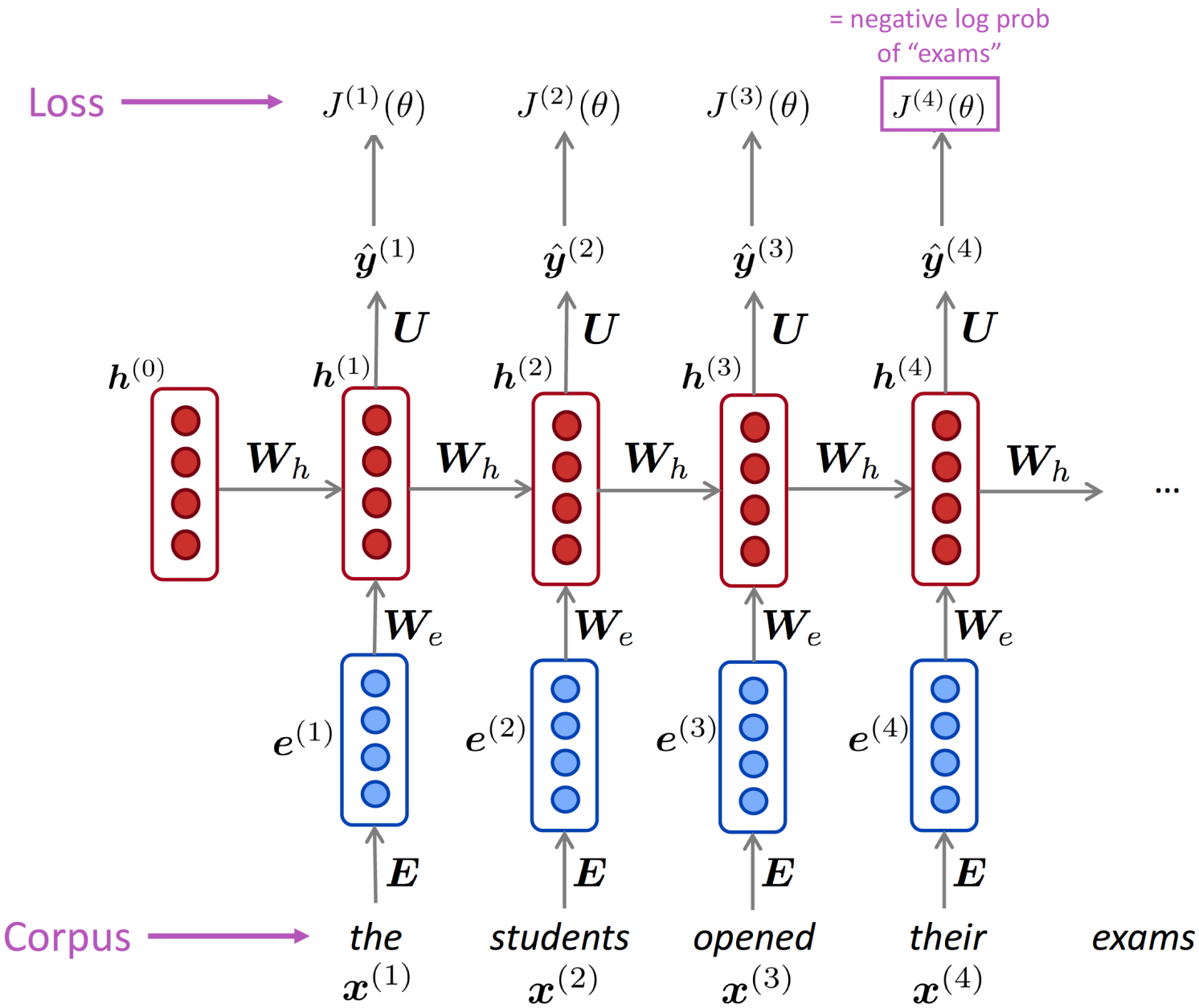
RNN LM: Training



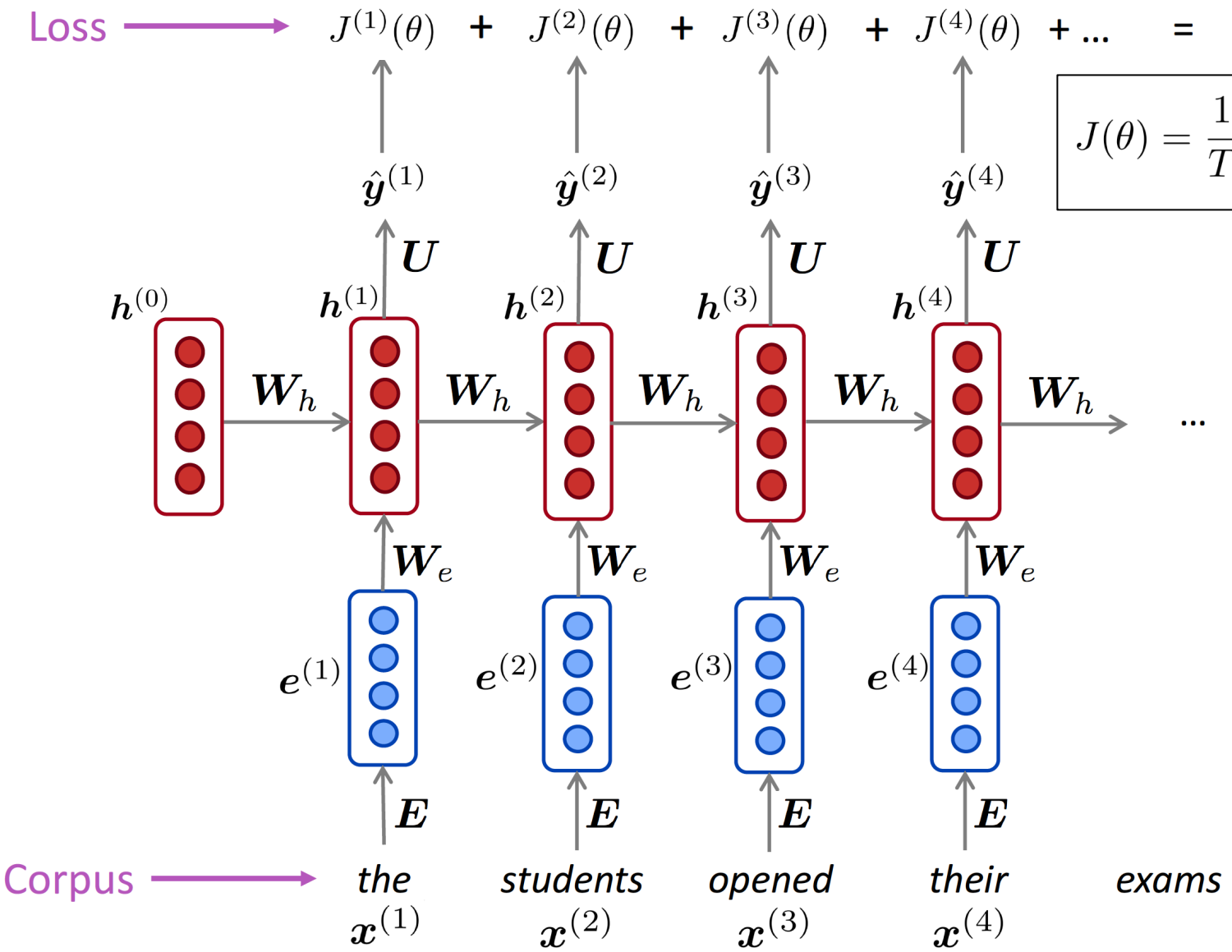
RNN LM: Training



RNN LM: Training



RNN LM: Training



RNN LM: Generating Text ☺

- Corpus: Obama speeches:

> *The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done.*

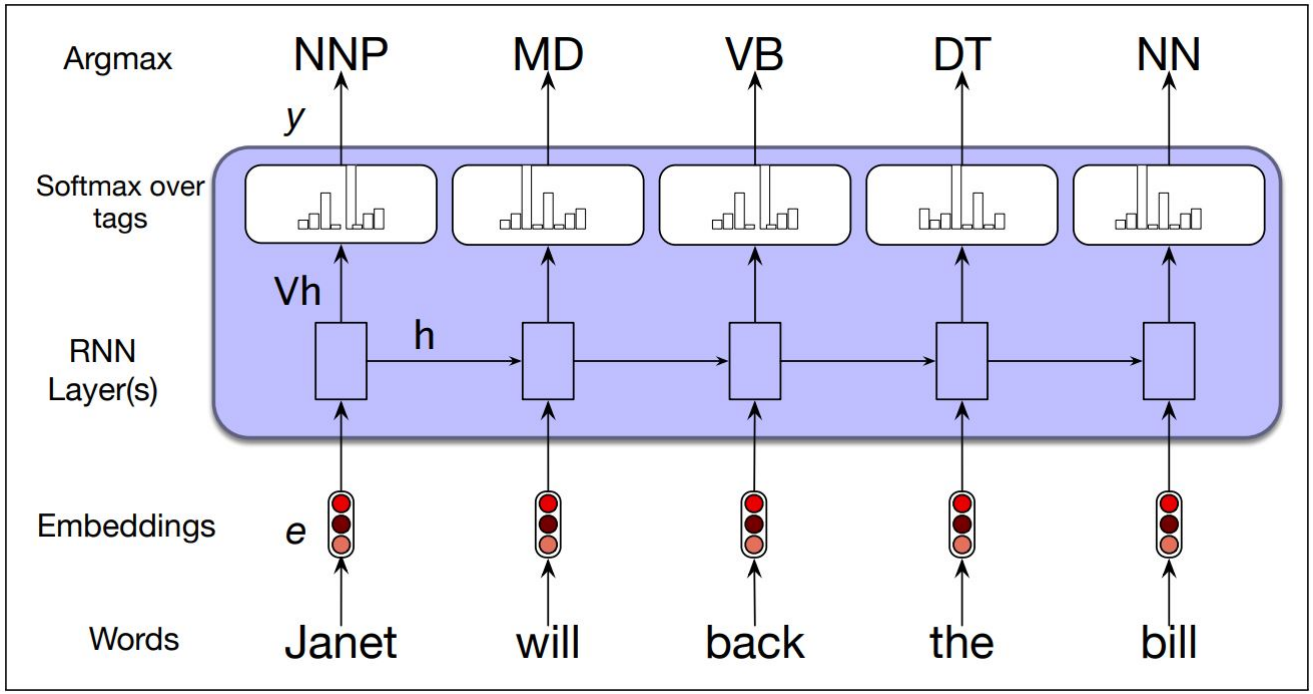
- Corpus: Harry Potter:

> "Sorry," Harry shouted, panicking—"I'll leave those brooms in London, are they?"
>
> "No idea," said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry's shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn't felt it seemed. He reached the teams too.

RNNs for Sequence Labeling

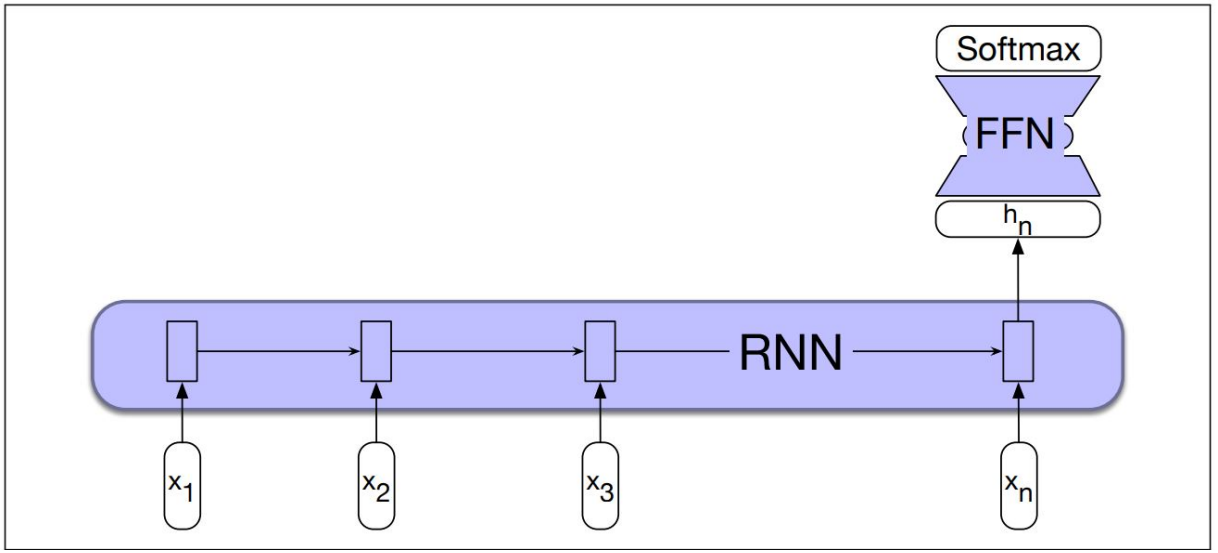
- Sequence labeling = assigning a label to each element of a sequence
- Ex: Part-of-speech tagging

- Inputs: word embeddings
- Outputs: tag probabilities



RNNs for Sequence Classification

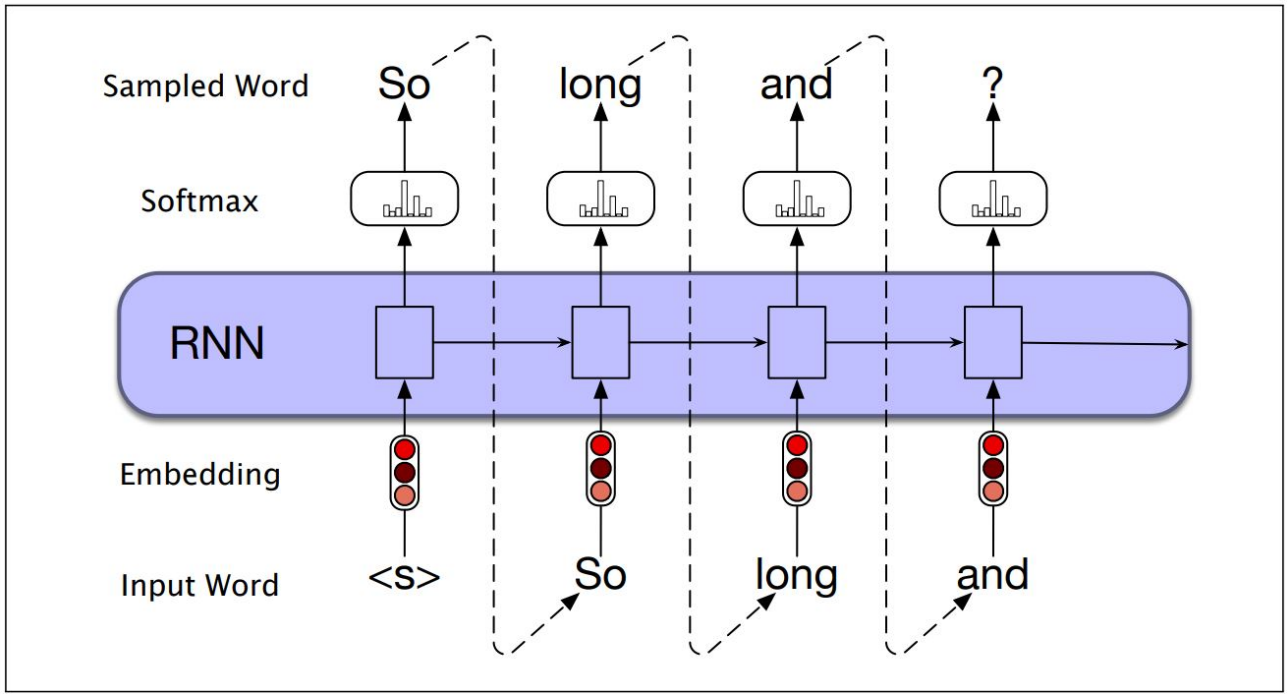
- Sequence classification = assigning a single label to a sequence of inputs
- examples: sentiment analysis, topic classification, spam detection
- here the last hidden layer h_n constitutes a compressed representation of the sequence. (alternative: pooling (e.g. mean) over $\{h_i\}$)



- no intermediate outputs = no loss terms for those elements
- loss is computed only on final task (end-to-end training)

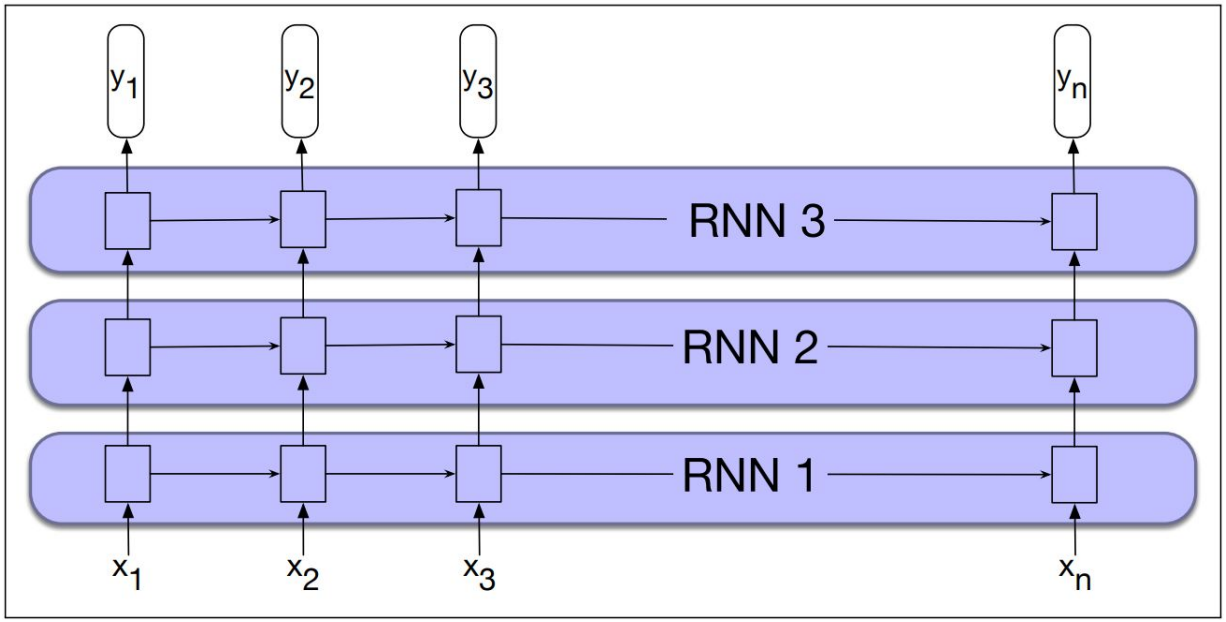
RNNs for Text Generation

- Used in: Question Answering, Machine Translation, Text Summarization



- Basic approach: Autoregressive generation
  - sample word by using beginning of sentence marker <s> as first input (or better: longer text as input)
  - use word embedding for first word and sample again
  - continue generating until </s> is reached or up to fixed length

Stacked and Bidirectional RNNs

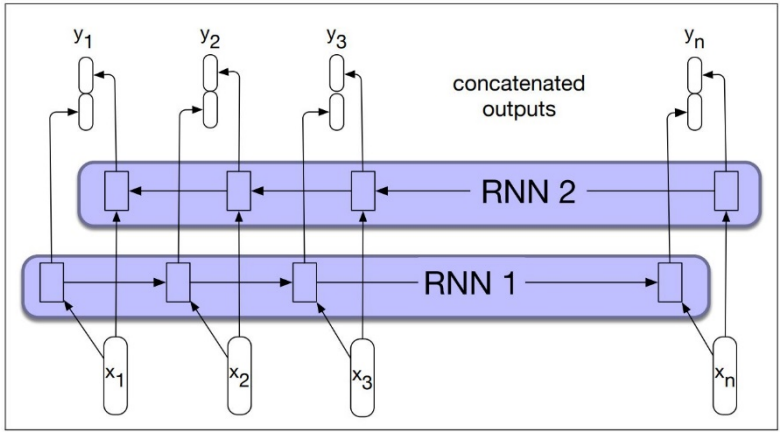


- Stacked RNN: Output of one layer serves as input to another
- Generally better performance than single-layer
- Subsequent stacked layers offer more abstraction

Stacked and Bidirectional RNNs

- Bidirectional RNN: Introduces right-context (looking into the future)
- Usual forward traversal (left-to-right):

$$\mathbf{h}_t^f = RNN_{forward}(\mathbf{x}_1, \ldots, \mathbf{x}_t)$$



- Adding the reversed input sequence

$$\mathbf{h}_t^b = RNN_{backward}(\mathbf{x}_t, \ldots \mathbf{x}_n)$$

- Concatenate both representations before prediction

$$\mathbf{h}_t = [\mathbf{h}_t^f ; \mathbf{h}_t^b]$$
$$= \mathbf{h}_t^f \oplus \mathbf{h}_t^b$$

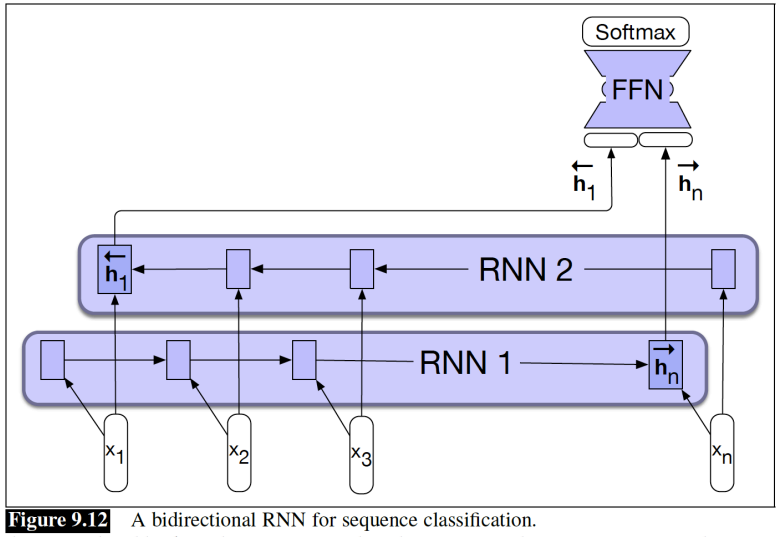
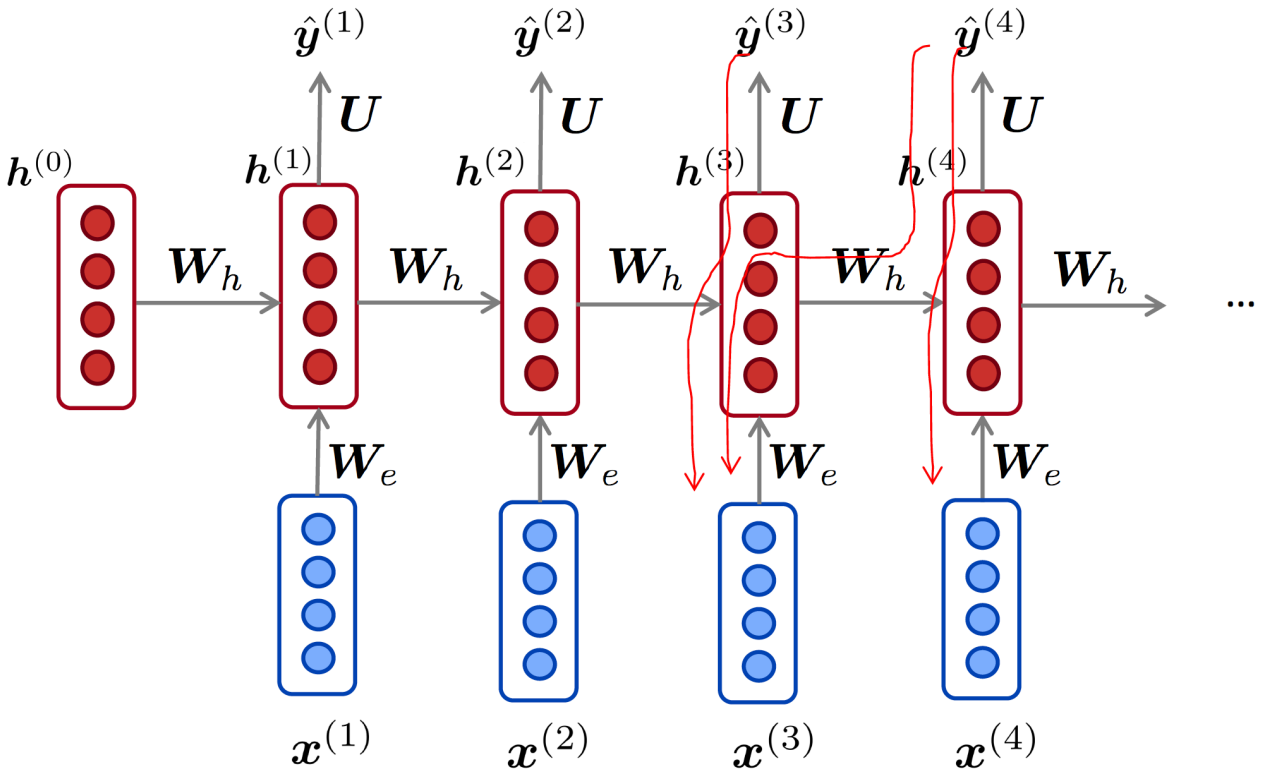


Figure 9.12 A bidirectional RNN for sequence classification.

RNN Problems

- Increasing distance is harder to represent
- Information encoded in one hidden state is mostly local, constantly overwritten by new information
- Distant information is critical for many language applications
- Especially in sequence classification tasks:
  - The final hidden state is a bottleneck for the learning process (backpropagation)
  - Long backpropagation can lead to vanishing gradients
  - Vanishing gradient: Repeated multiplications drive gradient to zero

Simple RNN: Vanishing Gradients



$$h^{(t)} = W_h f\left(h^{(t-1)}\right) + W_e x^{(t)}$$
$$\hat{y}^{(t)} = U\, f(h^{(t)})$$

$$h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}$$
$$\hat{y}_t = W^{(S)} f(h_t)$$

multiply same matrix at each BPTT step → possibly leads to vanishing gradient; problem: possibly important contributions to / from the past vanish

Simple RNN: Vanishing Gradients

$$
\begin{aligned}
h_t &= W f(h_{t-1}) + W^{(hx)} x_{[t]} \\
\hat{y}_t &= W^{(S)} f(h_t)
\end{aligned}
$$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

Simple RNN: Vanishing Gradients

$$
\begin{aligned}
h_t &= W f(h_{t-1}) + W^{(hx)} x_{[t]} \\
\hat{y}_t &= W^{(S)} f(h_t)
\end{aligned}
$$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

this may become very small (or very large)

Simple RNN: Vanishing Gradients

$$
\begin{aligned}
h_t &= W f(h_{t-1}) + W^{(hx)} x_{[t]} \\
\hat{y}_t &= W^{(S)} f(h_t)
\end{aligned}
$$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

this may become very small (or very large)

chain rule:

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

Simple RNN: Vanishing Gradients

$$
\begin{aligned}
h_t &= W f(h_{t-1}) + W^{(hx)} x_{[t]} \\
\hat{y}_t &= W^{(S)} f(h_t)
\end{aligned}
$$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

this may become very small (or very large)

chain rule: $$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|\mathrm{diag}[f'(h_{j-1})]\| := \beta_W \beta_h$$

Simple RNN: Vanishing Gradients

$$h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}$$
$$\hat{y}_t = W^{(S)} f(h_t)$$

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

this may become very small (or very large)

chain rule: $$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|\mathrm{diag}[f'(h_{j-1})]\| := \beta_W \beta_h$$

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_W \beta_h)^{t-k}$$

Simple RNN: Vanishing Gradients

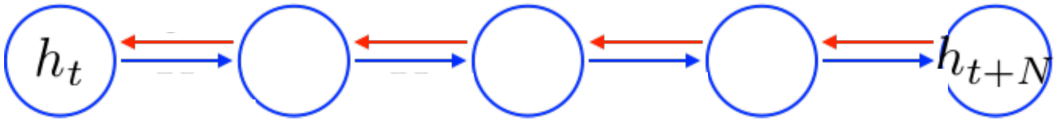
$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_W \beta_h)^{t-k}$$

- observe this influence from / to the past go to zero: →
  - either there IS no such dependency OR
  - there is a wrong choice of parameters / architecture cannot model this influence properly

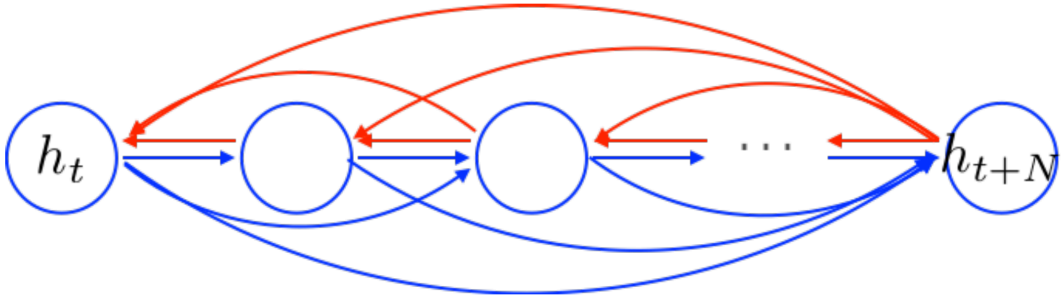
Contribution of Gated Nodes (GRU or LSTM)

- Simple RNN: $h_t = f\left(W^{(hh)} h_{t-1} + W^{(hx)} x_t\right)$

→ error must backpropagate through all intermediate nodes:



- idea: introduce shortcuts:

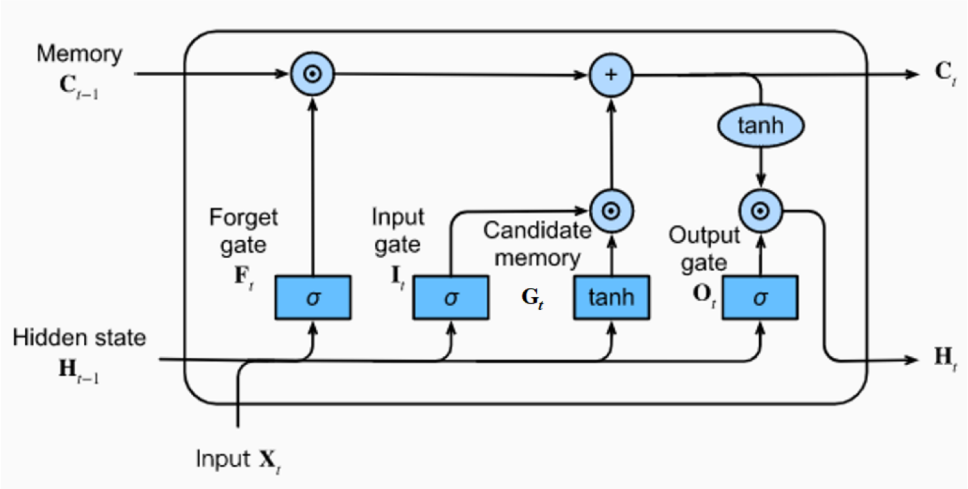


let gated units learn which shortcuts are necessary → 😊

even better: use FF (parallel input) + attention → 😊 😊 😊

Long Short-term Memory (LSTM)

- More complex RNN architectures to address these issues
- Long Short-term Memory (Hochreiter and Schmidhuber, 1997) is an RNN unit variant
- Divides the problem into
  - Removing unneeded information
  - Adding new information
  - Keeping separate states memory **C** and hidden state **H**
- Uses gates to control flow of information



Long Short-term Memory (LSTM)

- Gates:
  - Use additional weights **U** and **W**
  - Use input **X**, hidden layer **H**, memory **C**

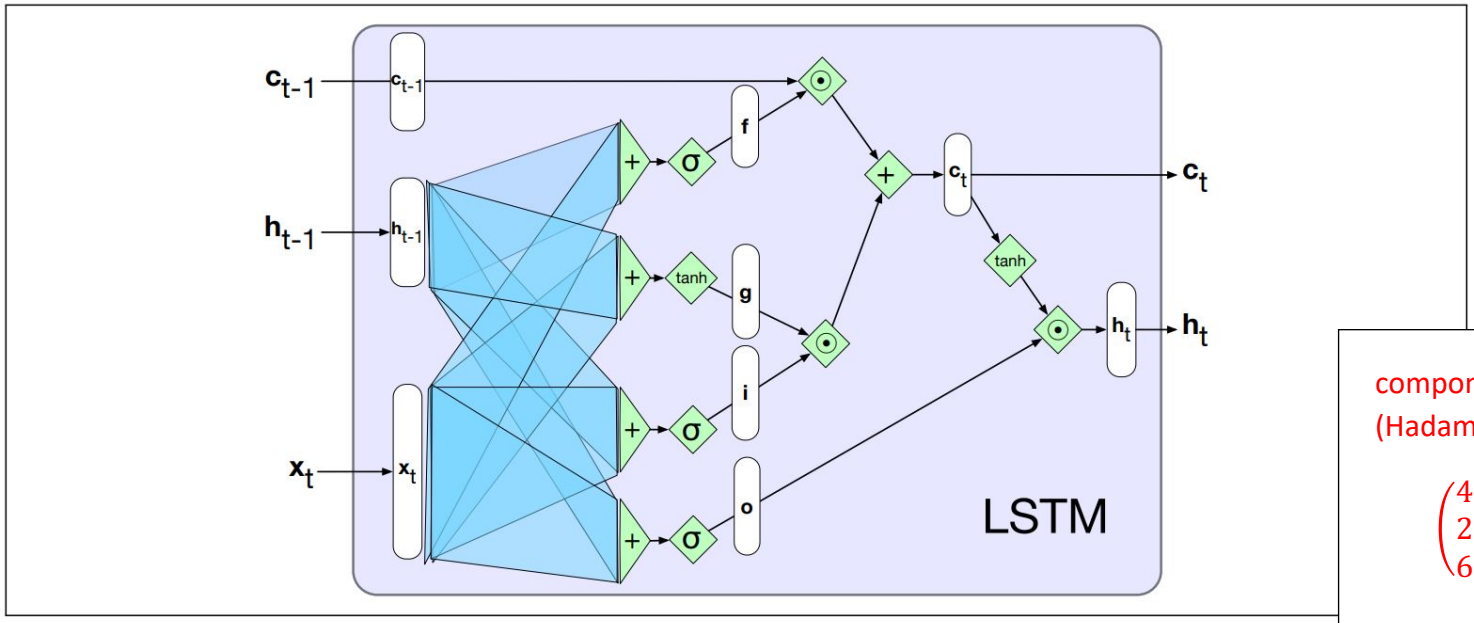
- Forget gate $\mathbf{f}_t = \sigma(\mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{W}_f \mathbf{x}_t)$

- Input gate $\mathbf{i}_t = \sigma(\mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{W}_i \mathbf{x}_t)$

- Output gate $\mathbf{o}_t = \sigma(\mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{W}_o \mathbf{x}_t)$

- Candidate memory (update candidate) $\mathbf{g}_t = \tanh(\mathbf{U}_g \mathbf{h}_{t-1} + \mathbf{W}_g \mathbf{x}_t)$

Long Short-term Memory (LSTM)



component-wise (Hadamard) product:

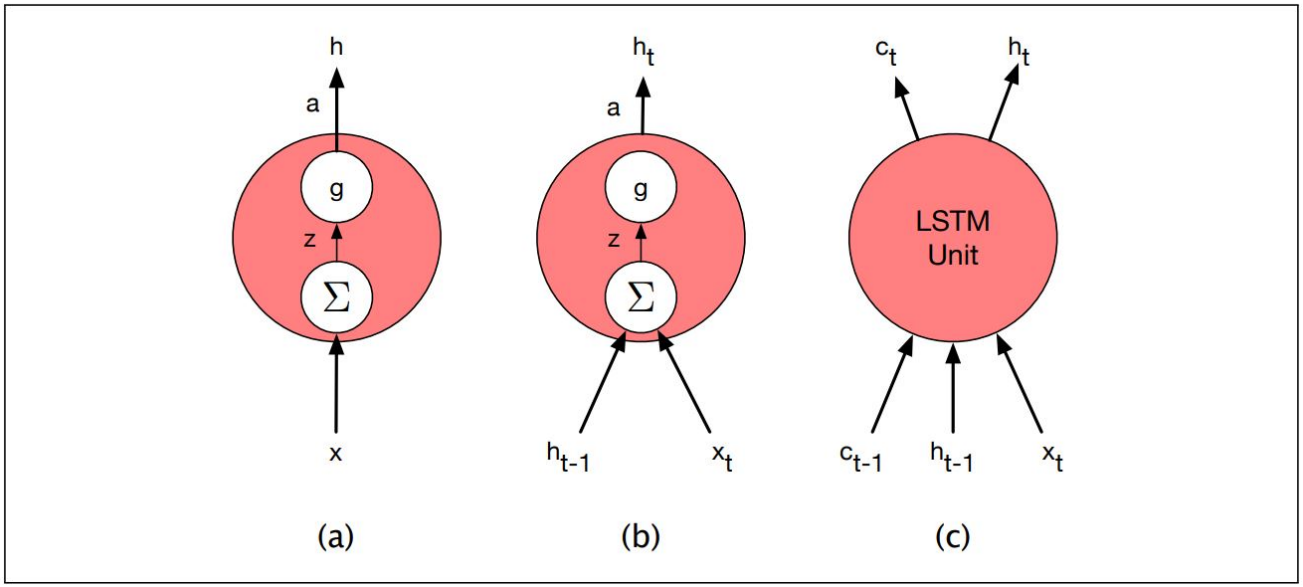
$$\begin{pmatrix}4\\2\\6\end{pmatrix} \odot \begin{pmatrix}1\\0\\2\end{pmatrix} = \begin{pmatrix}4\\0\\12\end{pmatrix}$$

Combining gates:

- forget gate controls how much of the previous memory to keep $\mathbf{k}_t = \mathbf{c}_{t-1} \odot \mathbf{f}_t$
- input gate controls how much of the proposed update to keep $\mathbf{j}_t = \mathbf{g}_t \odot \mathbf{i}_t$
- Add inputs to update-memory $\mathbf{c}_t = \mathbf{j}_t + \mathbf{k}_t$
- Combine current memory with outputs $\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$

Gated Units and Networks

- Evolution of processing units
  - (a) basic feedforward Neural Network
  - (b) RNN architecture with previous hidden state
  - (c) LSTM architecture with previous hidden state and memory



- All units are modular → gated units
- Can be unrolled into a purely feedforward network

Self-Attention Networks

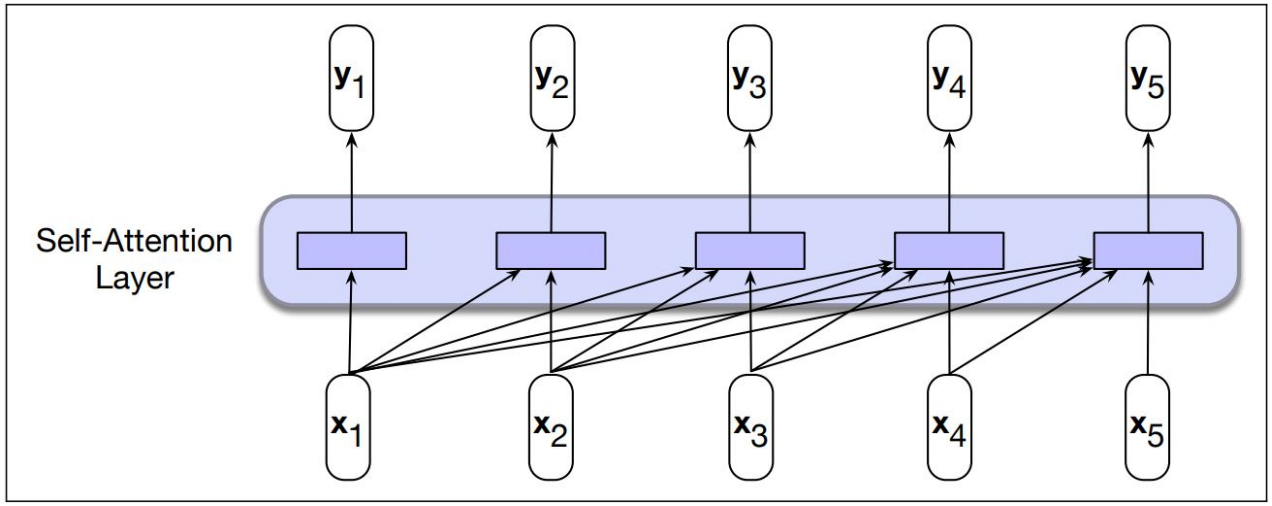
- Gates allow more distant information flow
- Core problem remains:
  - Series of recurrent connections leads to information loss + difficulties training
- Further computational problems:
  - Recurrent networks make parallelization hard

- Solution: Transformer networks
  - Mapping sequence of inputs to sequence of outputs

- Underlying mechanism: Self-attention

Self-Attention Networks

- Self-attention layer:
  - Extract information from arbitrary length context
  - No passing through intermediary recurrent connections



- For each input: Access to all information from previous inputs = availability of long term context
- Computation per node is independent = parallelizable

Self-Attention Networks

Basic attention mechanism: Scoring

- Comparing elements within a sequence in view of their relevance *for current computation*

  example: Computing y_3 = compare input x_3 to x_1 , x_2 and x_3

- Simple comparison: dot product

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$$

  result: Scalar from $-\infty$ to ∞

- Normalize scores into a weight vector a_{ij} using Softmax

$$\begin{aligned} \alpha_{ij} &= \text{softmax}(\text{score}(\mathbf{x}_i, \mathbf{x}_j)) \ \forall j \leq i \\ &= \frac{\exp(\text{score}(\mathbf{x}_i, \mathbf{x}_j))}{\sum_{k=1}^{i} \exp(\text{score}(\mathbf{x}_i, \mathbf{x}_k))} \ \forall j \leq i \end{aligned}$$

- Generate output by summing inputs weighed by score

$$\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{x}_j$$

Self-Attention for Transformers

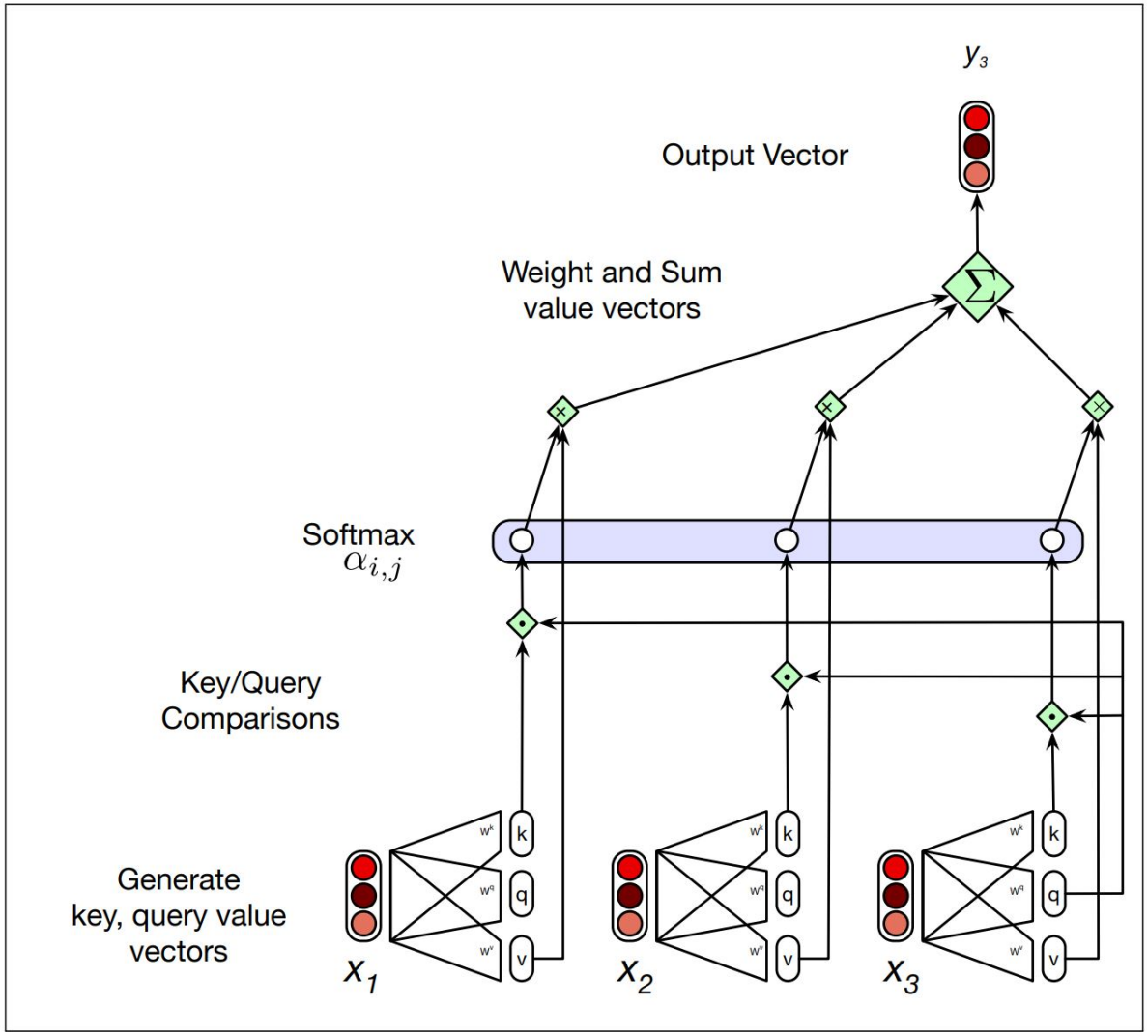
- Transformers change the way in which different inputs contribute to the output of the self-attention network
- How each input token can act in the attention process:
  - as Query: current focus of attention
  - as Key: preceding input being compared to the query
  - as Value: used to compute the output
- Transformers use weight matrices W^Q, W^K, W^V (all $\in \mathbb{R}^{d \times d}$) to project inputs to these roles

$$\mathbf{q}_i = \mathbf{W}^{\mathbf{Q}}\mathbf{x}_i; \quad \mathbf{k}_i = \mathbf{W}^{\mathbf{K}}\mathbf{x}_i; \quad \mathbf{v}_i = \mathbf{W}^{\mathbf{V}}\mathbf{x}_i$$

- Using projections, scoring becomes $\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{q}_i \cdot \mathbf{k}_j$
- Output calculation is done via the values $\mathbf{y}_i = \sum_{j \leq i} \alpha_{ij} \mathbf{v}_j$

Self-Attention for Transformers

- Transformer-style self-attention computation for a single output y_3



Properties of Self-Attention

- Since dot products can become too large, they are instead scaled by the square root of the dimensionality of the key vector

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

- Key property of this computation:
  - Each output y_i is computed independently
  - Can be further sped up through matrix multiplication (**X, Q, K, V** all $\in \mathbb{R}^{N \times d}$)

$$\mathbf{Q} = \mathbf{X}\mathbf{W}^{\mathbf{Q}}; \ \mathbf{K} = \mathbf{X}\mathbf{W}^{\mathbf{K}}; \ \mathbf{V} = \mathbf{X}\mathbf{W}^{\mathbf{V}}$$

$$\textit{SelfAttention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\intercal}}{\sqrt{d_k}}\right)\mathbf{V}$$

- Note: Attention is quadratic w.r.t. length of the input

Masking

$$SelfAttention(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}\left(\frac{\mathbf{QK}^\intercal}{\sqrt{d_k}}\right)\mathbf{V}$$

- Problem for decoder style applications (e.g. language modelling): via $\mathbf{QK}^\intercal$ each query would be compared with each key, including those that follow the query
- → we see what we are predicting → solution: masking

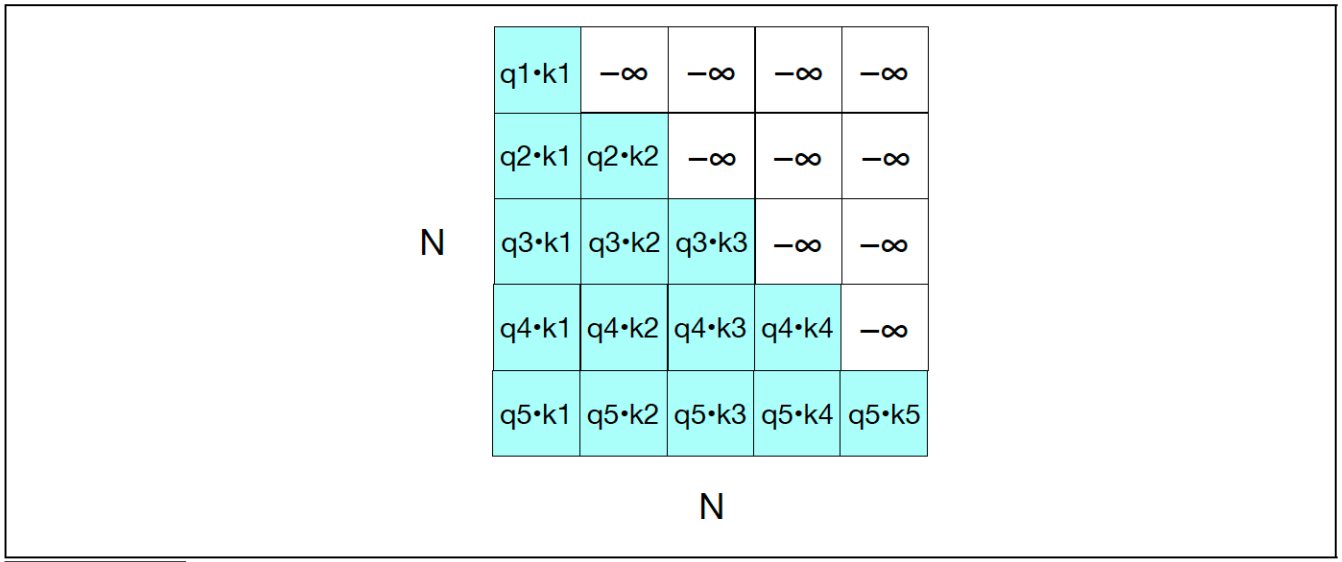
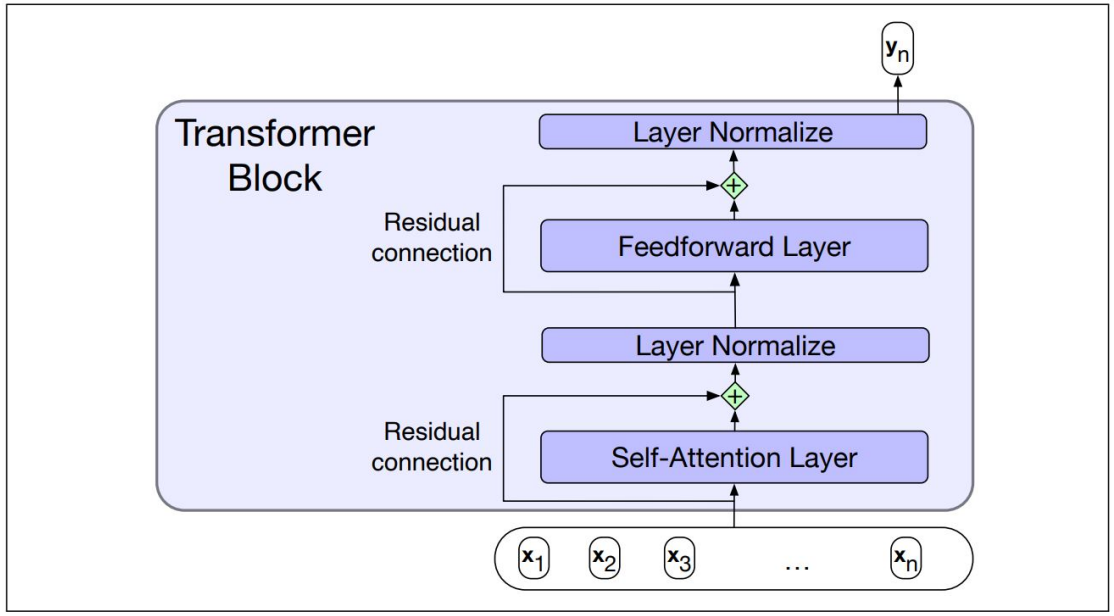


Figure 9.17 The $N \times N$ $\mathbf{QT}^\intercal$ matrix showing the $q_i \cdot k_j$ values, with the upper-triangle portion of the comparisons matrix zeroed out (set to $-\infty$, which the softmax will turn to zero).

Transformer Blocks

- Basis of the transformer block: Self-attention layer
- Additional features:
  - Feedforward layers
  - Residual connections
  - Normalizing layers

residual connection is impoartant because you want each layer to learn something. So if you dont add embedding will not represent to input maybe distribution changed to much



- Transformer blocks have identical input and output dimensions, meaning they can be stacked

Layer Norm

- Layer Norms are used to improve training performance
- Keeps values in hidden layers within a specific range to facilitate gradient-based training
- Based on standard z-score via computing mean μ and std deviation σ

$$\mu = \frac{1}{d_h}\sum_{i=1}^{d_h} x_i$$

$$\sigma = \sqrt{\frac{1}{d_h}\sum_{i=1}^{d_h}(x_i-\mu)^2}$$

- Component normalization $\hat{\mathbf{x}} = \frac{(\mathbf{x}-\mu)}{\sigma}$

- Optional: Learnable parameters for gain and offset

$$LayerNorm = \gamma\,\hat{\mathbf{x}} + \beta$$

Multihead Attention

- Words can relate to each other in different ways at the same time
- hard to learn this with a single transformer block

- Solution:

bigger is better :D

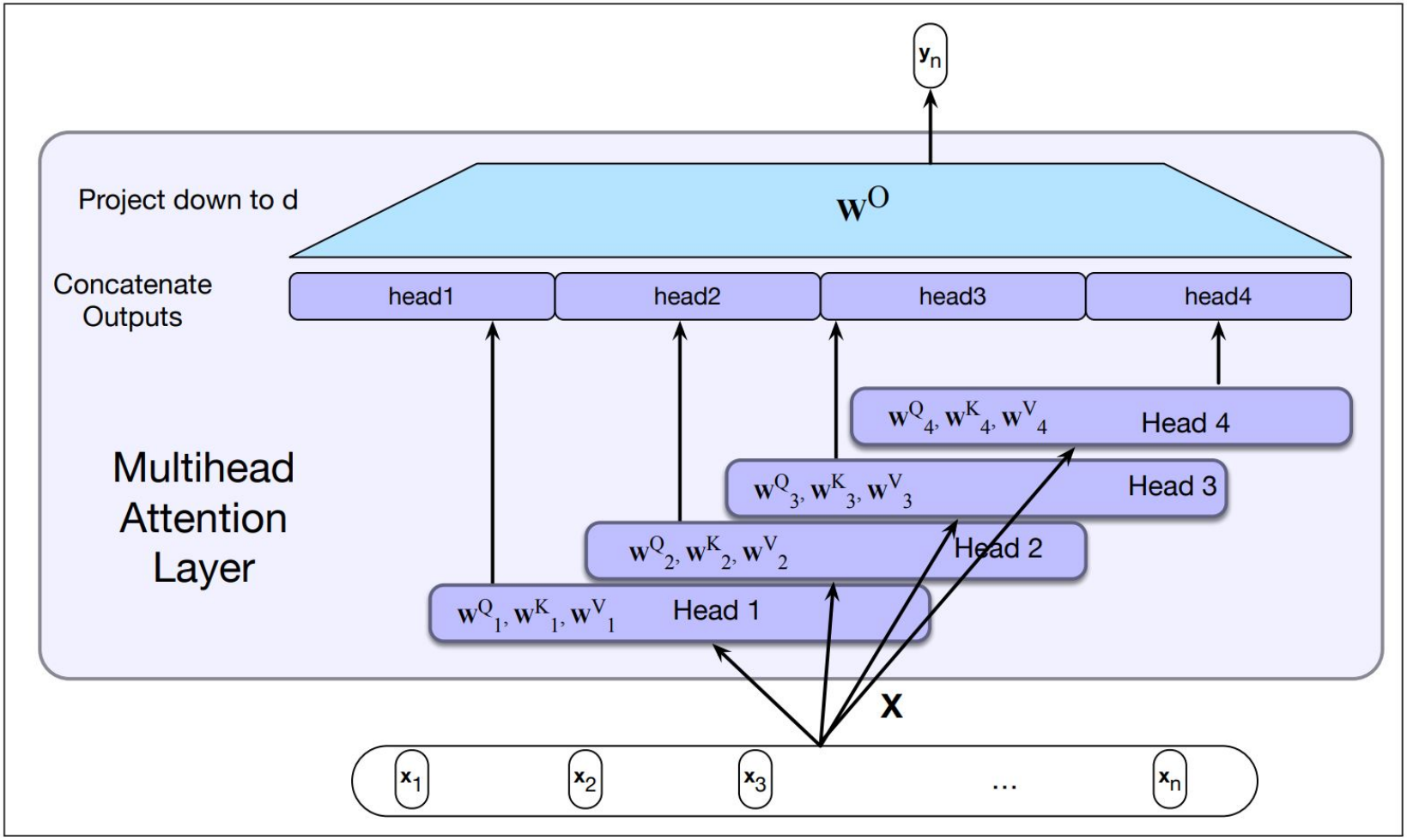
  - Multihead self-attention layers
  - each layer features a distinct set of parameters and weights
  - input is copied across all heads
  - computed in parallel at the same depth of the model
  - at the end of a block reduced back to the original dimensionality by concatenation and linear projection via weight matrix W^0

$$MultiHeadAttn(\mathbf{X}) = (\mathbf{head}_1 \oplus \mathbf{head}_2 \ldots \oplus \mathbf{head}_h)\mathbf{W}^O$$

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_i^Q \; ; \; \mathbf{K} = \mathbf{X}\mathbf{W}_i^K \; ; \; \mathbf{V} = \mathbf{X}\mathbf{W}_i^V$$

$$\mathbf{head}_i = SelfAttention(\mathbf{Q}, \mathbf{K}, \mathbf{V})$$

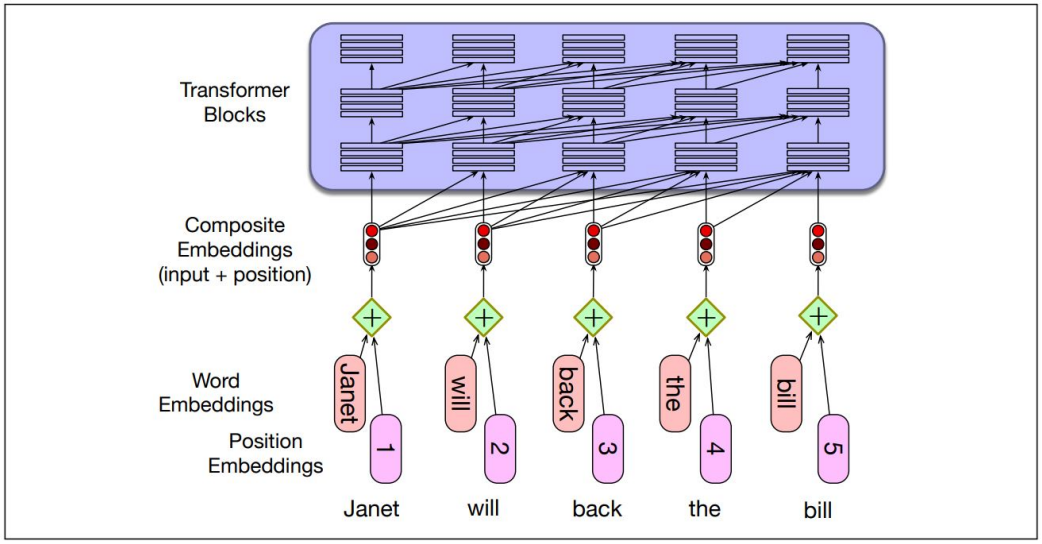
Multihead Attention



Positional Encodings for Transformers

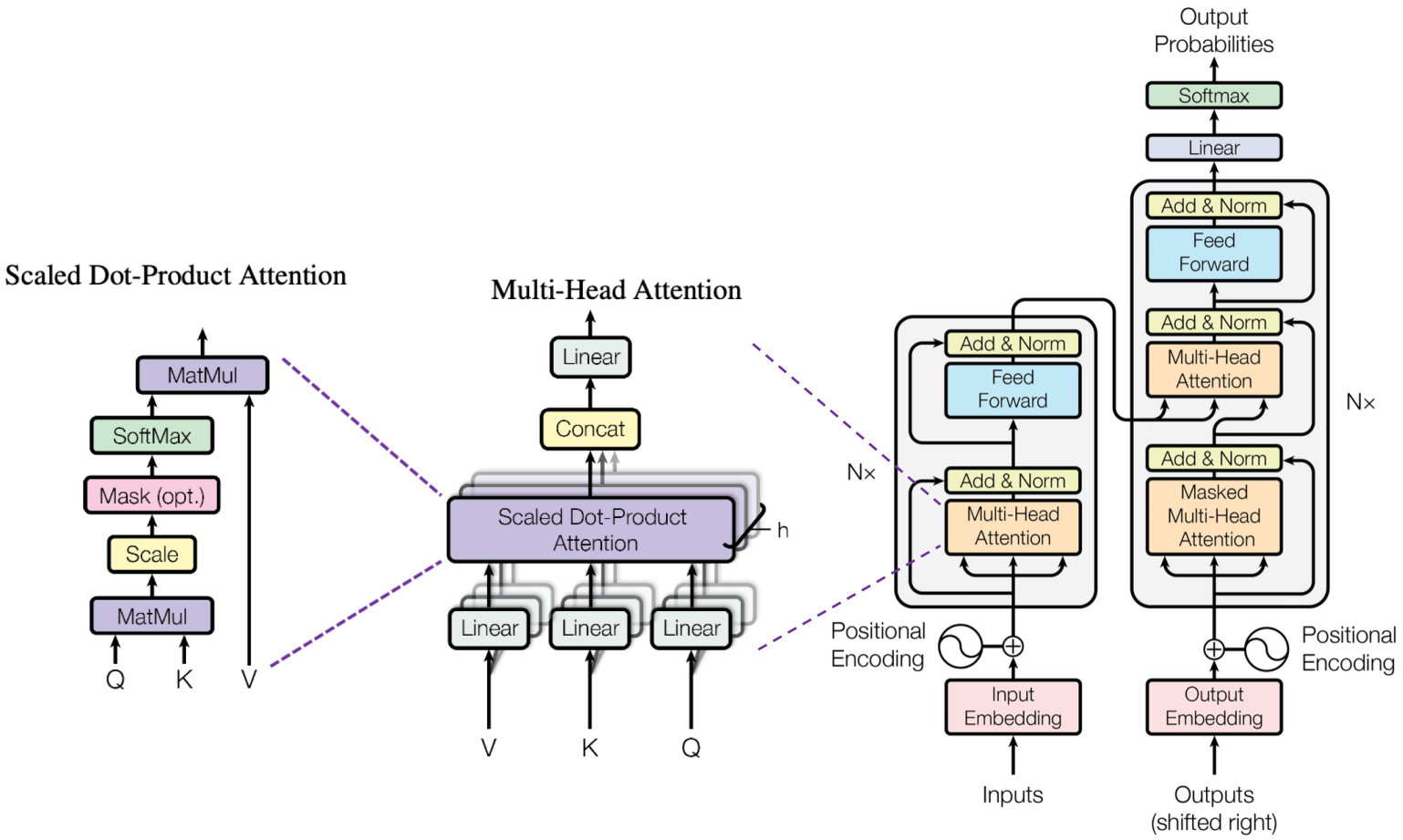
- Positions of tokens are built-in with RNNs
- This is not the case with transformers
- Problem: No notion of relative or absolute position in a sequence

- Solution: Modify input embeddings with positional encodings
- Example: Randomly initialized, trainable positional encodings
  - Either use absolute positions or a mapping to relative positions



The Transformer

- Full schema of the original Transformer architecture (Vaswani et al., 2017) for NMT (seq2seq / Encoder-Decoder)



Transformers as Language Models

- Using transformers we can train a language model the same way as with RNNs:
  - Semi-supervised learning
  - Training corpus of plain text
  - Predicting the next word using teacher forcing
  - Compute loss via cross-entropy over the sequence

- Advantages:
  - Calculations are no longer serial
  - Parallelization possible!
  - Resulting model can be evaluated via perplexity
  - New text can be autoregressively generated

- Other possible tasks: Text completion, question answering

Contextual Generation

Contextual generation: model has access to full priming context & all own subsequently generated outputs so far

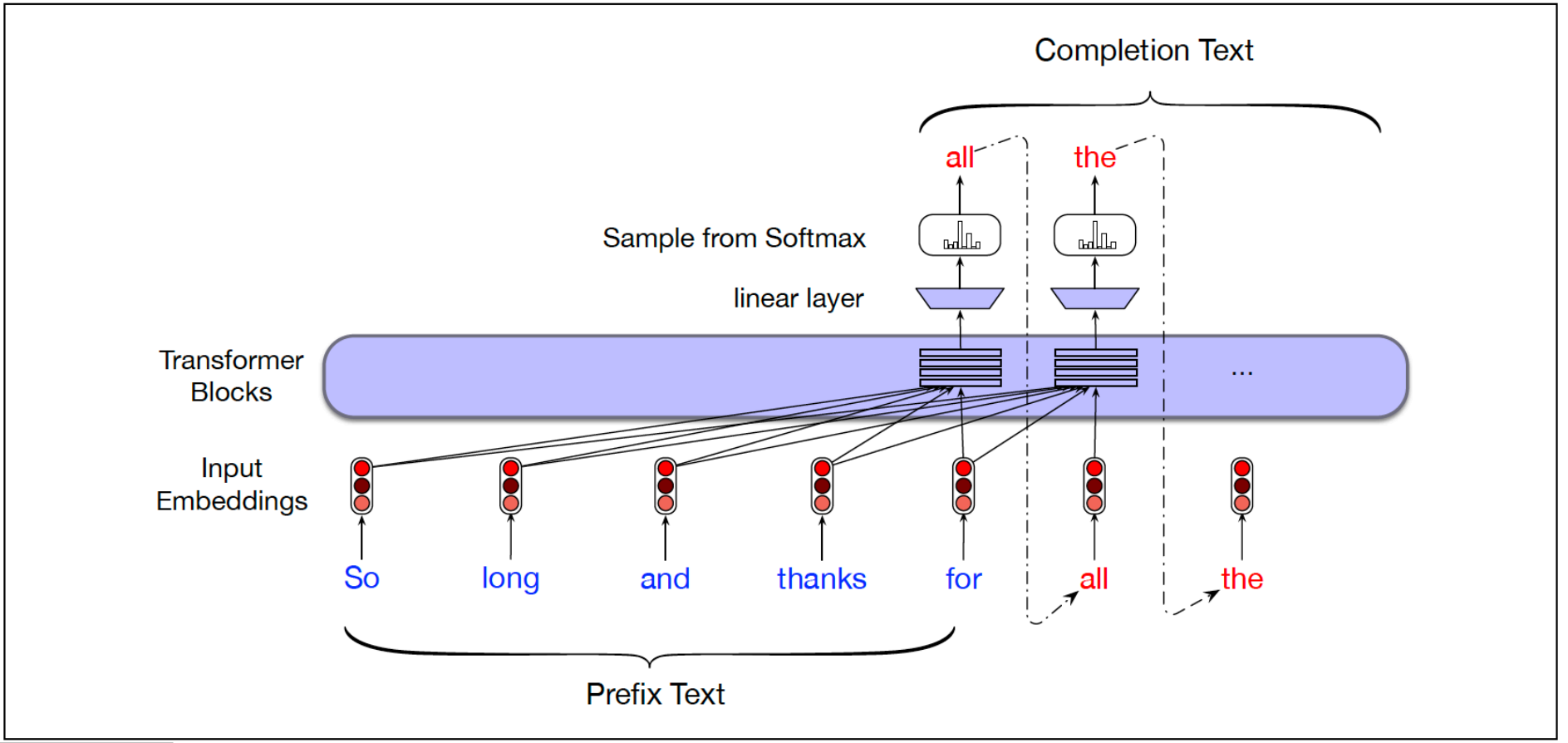
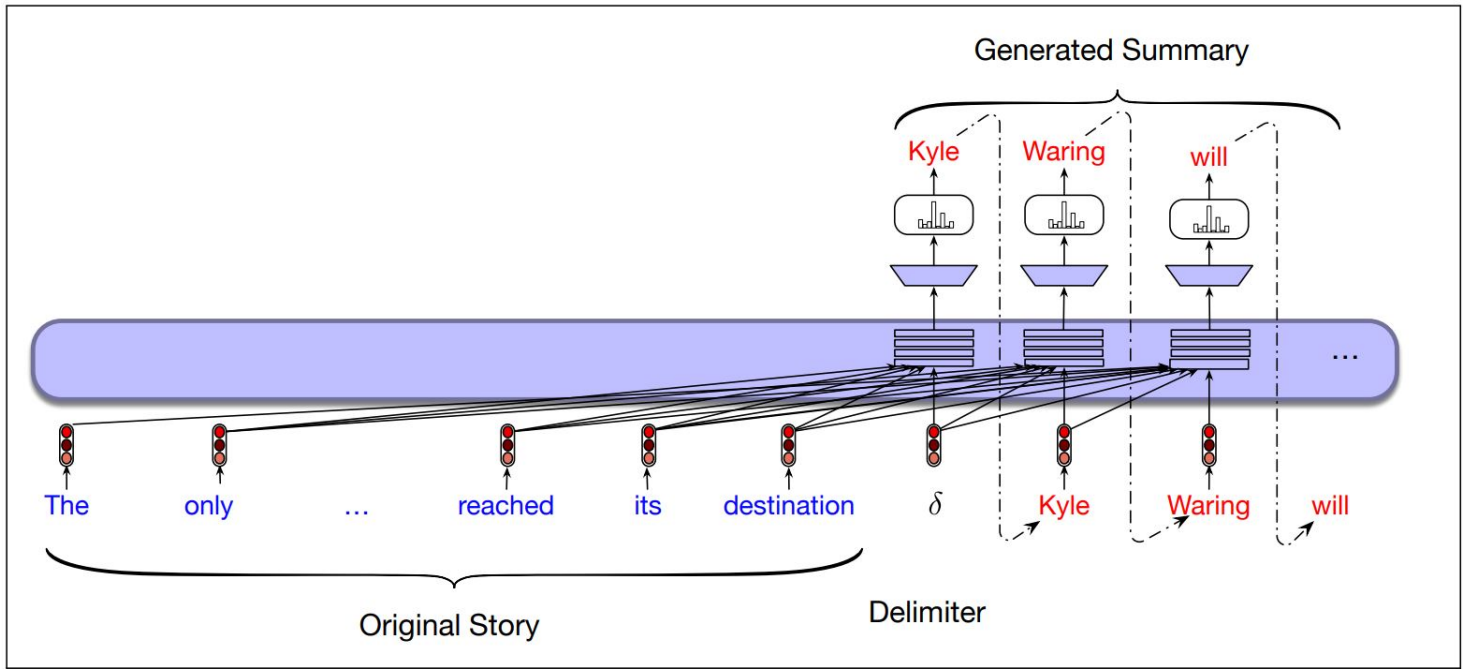


Figure 9.22 Autoregressive text completion with transformers.

Text Summarization using Transformers

- Autoregressive models can be trained for summarization tasks
- Process:
  - Append full length text with its summary, separated by a unique marker
  - Train an autoregressive language model using teacher forcing



Text Summarization using Transformers

Original Article

The only thing crazier than a guy in snowbound Massachusetts boxing up the powdery white stuff and offering it for sale online? People are actually buying it. For $89, self-styled entrepreneur Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says.

But not if you live in New England or surrounding states. "We will not ship snow to any states in the northeast!" says Waring's website, ShipSnowYo.com. "We're in the business of expunging snow!"

His website and social media accounts claim to have filled more than 133 orders for snow – more than 30 on Tuesday alone, his busiest day yet. With more than 45 total inches, Boston has set a record this winter for the snowiest month in its history. Most residents see the huge piles of snow choking their yards and sidewalks as a nuisance, but Waring saw an opportunity.

According to Boston.com, it all started a few weeks ago, when Waring and his wife were shoveling deep snow from their yard in Manchester-by-the-Sea, a coastal suburb north of Boston. He joked about shipping the stuff to friends and family in warmer states, and an idea was born. His business slogan: "Our nightmare is your dream!" At first, ShipSnowYo sold snow packed into empty 16.9-ounce water bottles for $19.99, but the snow usually melted before it reached its destination...

Summary

Kyle Waring will ship you 6 pounds of Boston-area snow in an insulated Styrofoam box – enough for 10 to 15 snowballs, he says. But not if you live in New England or surrounding states.

Applying Transformers to Other Tasks

- Other possible transformer tasks include:
  - Sequence labeling
  - Sequence classification

- Usual way:
  - No direct training of the transformer on the task
  - Use pretrained transformer
  - Add feedforward layer on top to finetune on the task

Bibliography

(1) Dan Jurafsky and James Martin: Speech and Language Processing (3rd ed. draft, version Jan, 2022); Online: https://web.stanford.edu/~jurafsky/slp3/ (URL, Oct 2022) (this slideset is especially based on chapter 9)

(2) Chris Manning et al: "CS224n: Natural Language Processing with Deep Learning", Lecture Materials (slides and links to background reading) http://web.stanford.edu/class/cs224n/ (URL, Oct 2022), 2020

(3) https://www.youtube.com/playlist?list=PLoROMvodv4rOSH4v6133s9LFPRHjEmbmJ (URL, Oct 2022) (in [2])

(4) Rico Sennrich (University of Edinburgh): Neural Machine Translation: Breaking the Performance Plateau, Talk July 2016; http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf (URL, Aug 2018) (in [2])

(5) Richard Socher et al: "CS224n: Natural Language Processing with Deep Learning", Lecture Materials (slides and links to background reading) http://web.stanford.edu/class/cs224n/ (URL, May 2018), 2018

Recommendations for Studying

- minimal approach:

  work with the slides and understand their contents! Think beyond instead of merely memorizing the contents

- standard approach:

  minimal approach + read the corresponding pages in Jurafsky [1]

- interested students

  == standard approach + read the additional papers in bibliography. Download an available IPython implementation of Transformers and experiment with it, visualizing self attention scores.