Machine Translation

Machine Translation: a Natural Extension of Neural Language Modeling

You have already learned how to build it.

Machine Translation

- Input: a sentence written in a source language L_S
- Output: a corresponding sentence in a target language L_T
- Problem statement:
  - Supervised learning: given the input sentence, output its translation
  - Compute the conditional distribution over all possible translation given the input

    $$p(Y = (y_1, \ldots, y_T) | X = (x_1, \ldots, x_{T'}))$$

- *We have already learned every necessary ingredient for building a full neural machine translation system.*

Token Representation – One-hot Vectors

1. Build source and target vocabularies of unique tokens
   - For each of source and target languages,
     1. Tokenize: separate punctuations, normalize punctuations, …
        e.g., “I’m going” => (“I”, “’m”, “going”), replace ‘, ’, `, into “`”, …
        use Spacy.io, NLTK or Moses’ tokenizer.
     2. Subword segmentation: segment each token into a sequence of subwords
        e.g., “going” => (“go”, “ing”), use BPE [Sennrich et al., 2015]
     3. Collect all unique subwords, sort them by their frequencies (descending) and assign indices.
2. Transform each subword token into a corresponding one-hot vector.*

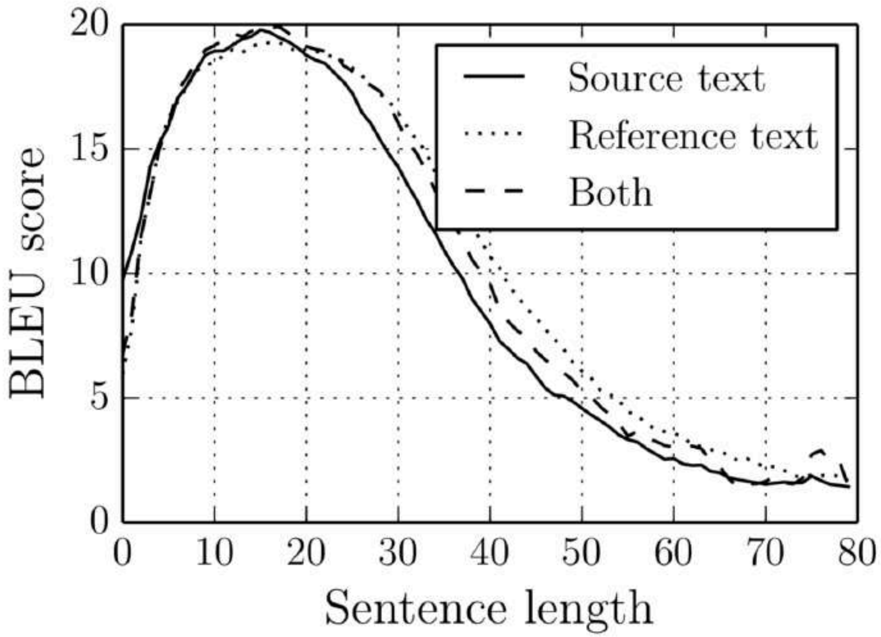
* Of course, don’t do it offline. Transform them online.

Encoder – Source Sentence Representation

- Encode the source sentence into a set of sentence representation vectors
  - # of encoded vectors is proportional to the source sentence length: often same. $H = (h_1, \ldots, h_{T'})$
  - Recurrent networks have been widely used [Cho et al., 2014; Sutskever et al., 2014], but CNN [Gehring et al., 2017; Kalchbrenner&Blunsom, 2013] and self-attention [Vaswani et al., 2017] are used increasingly more often. See Lecture 2 for details.
- We do not want to collapse them into a single vector.
  - Collapsing often corresponds to information loss.
  - Increasingly more difficult to encode the entire source sentence into a single vector, as the sentence length increases [Cho et al., 2014b].
  - We didn't know initially until [Bahdanau et al., 2015].

Encoder – Source Sentence Representation

- Encode the source sentence into a set of sentence representation vectors
- We do not want to collapse them into a single vector.
  - Increasingly more difficult to encode the entire source sentence into a single vector, as the sentence length increases [Cho et al., 2014b].



Encoder – Source Sentence Representation

- Encode the source sentence into a set of sentence representation vectors
- We do not want to collapse them into a single vector.
  - Increasingly more difficult to encode the entire source sentence into a single vector, as the sentence length increases [Cho et al., 2014b].
  - When collapsed, the system fails to translate a long sentence correctly.
    - **Source**: *An admitting privilege is the right of a doctor to admit a patient to a hospital or a medical centre <u>to carry out a diagnosis or a procedure, based on his status as a health care worker at a hospital.</u>*
    - **When collapsed:** *Un privilège d'admission est le droit d'un médecin de reconnaître un patient à l'hôpital ou un centre médical <u>d'un diagnostic ou de prendre un diagnostic en fonction de son état de santé.</u>*
  - The system translates reasonable up to a certain point, but starts drifting away.

Decoder – Language Modelling

- Autoregressive Language modelling with an infinite context $n \rightarrow \infty$
  - Larger context is necessary to generate a coherent sentence.
    - Semantics could be largely provided by the source sentence,
      but syntactic properties need to be handled by the language model directly.
  - Recurrent networks, self-attention and (dilated) convolutional networks
    - Causal structure must be followed.
    - See Lecture 3.
- **Conditional** Language modelling
  - The context based on which the next token is predicted is **two-fold**

$$p(Y|X) = \prod_{t=1}^{T} p(y_t | y_{<t}, X)$$

Decoder – Conditional Language Modelling

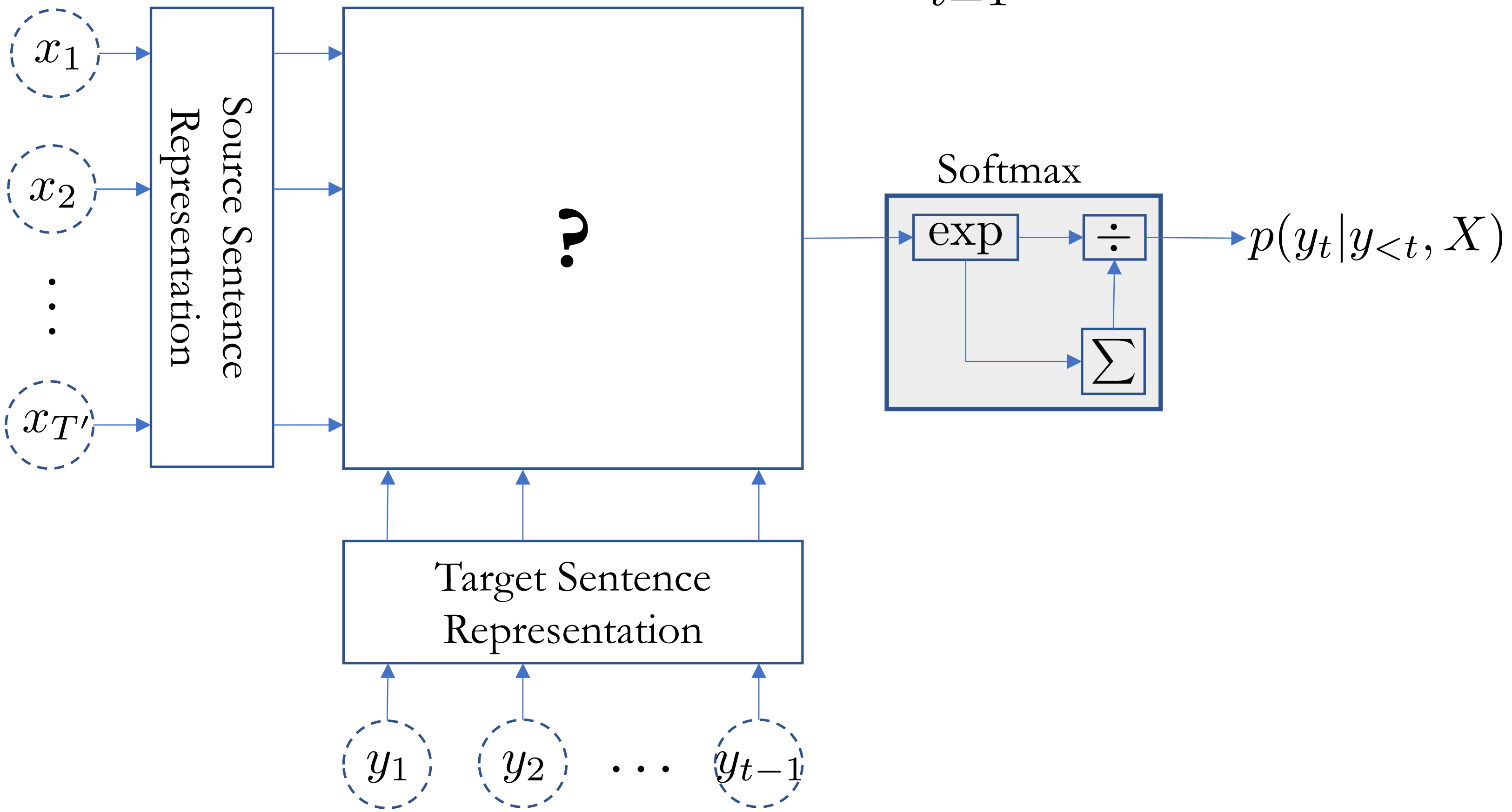
- Conditional Language modelling
  - The context based on which the next token is predicted is **two-fold**.

$$p(Y|X) = \prod_{t=1}^{T} p(y_t|y_{<t}, X)$$

- Supervised learning: T input-output training pairs per sentence
  - Input: the entire source sentence X and the preceding target tokens $y_{<t}$
  - Output: the next token y_t

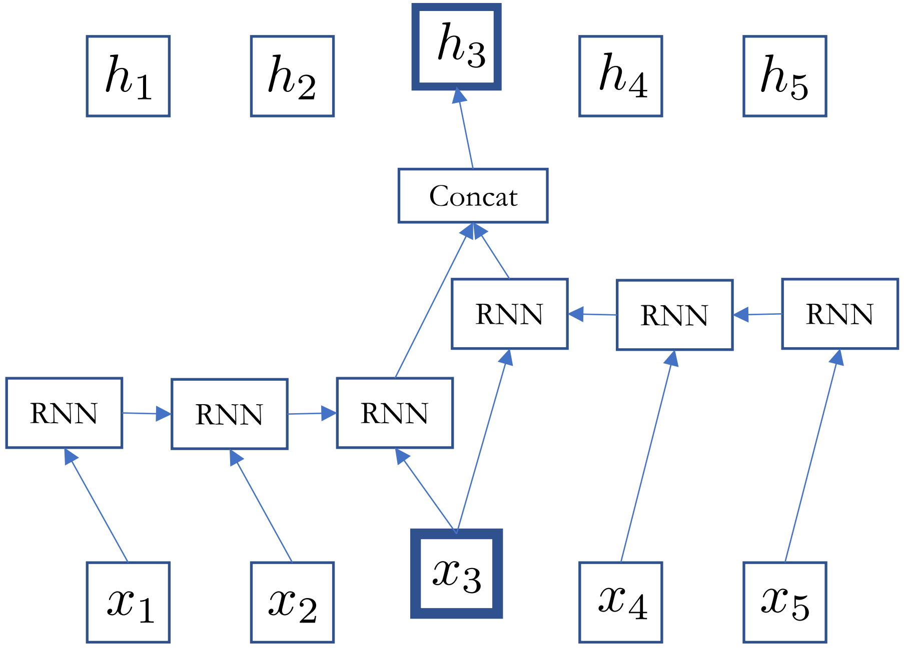
Decoder – Conditional Language Modelling

- Conditional Language modelling $p(Y|X) = \prod_{t=1}^{T} p(y_t|y_{<t}, X)$

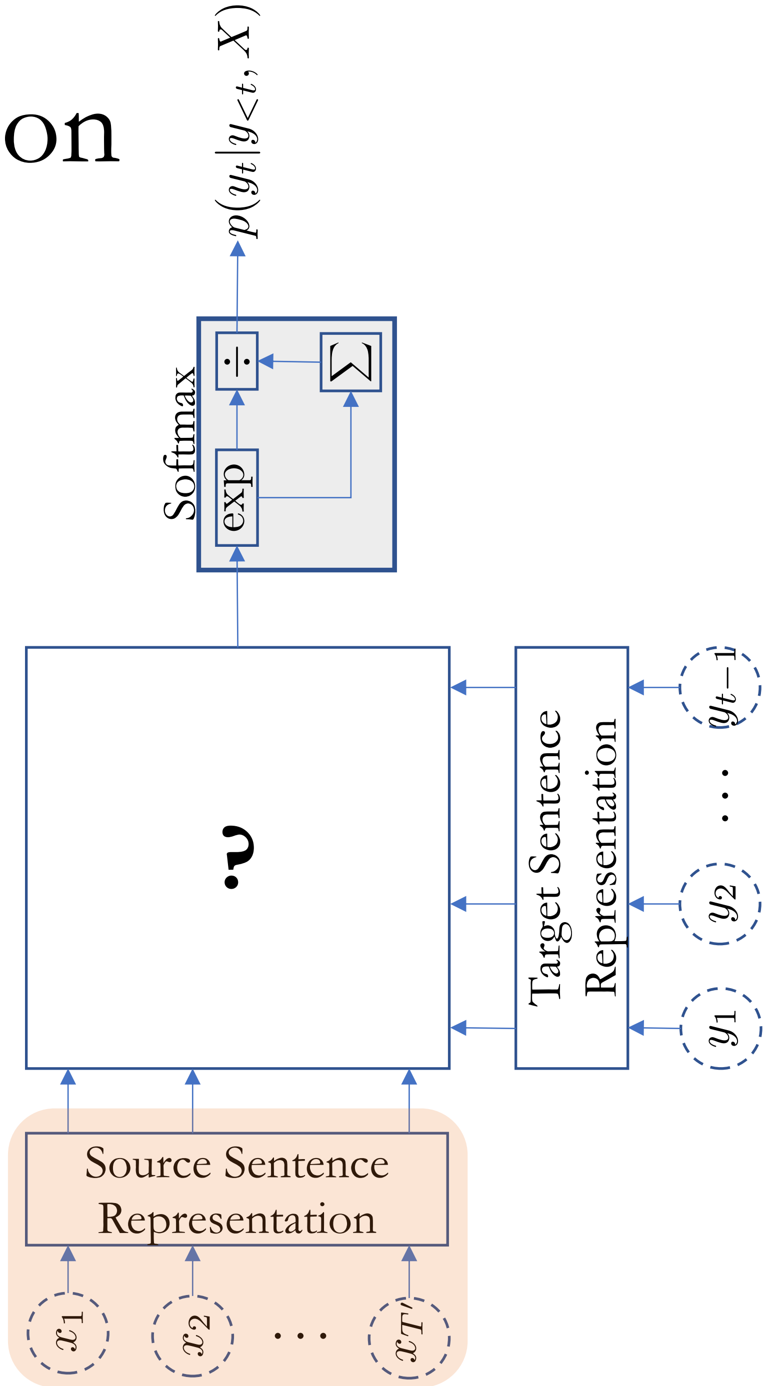


RNN Neural Machine Translation [Bahdanau et al., 2015]

1. Source sentence representation
   - A stack of bidirectional RNN's

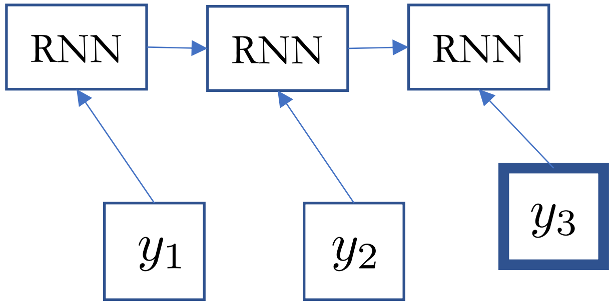


   - The extracted vector at each location is a **context-dependent vector representation.**



RNN Neural Machine Translation
[Bahdanau et al., 2015]

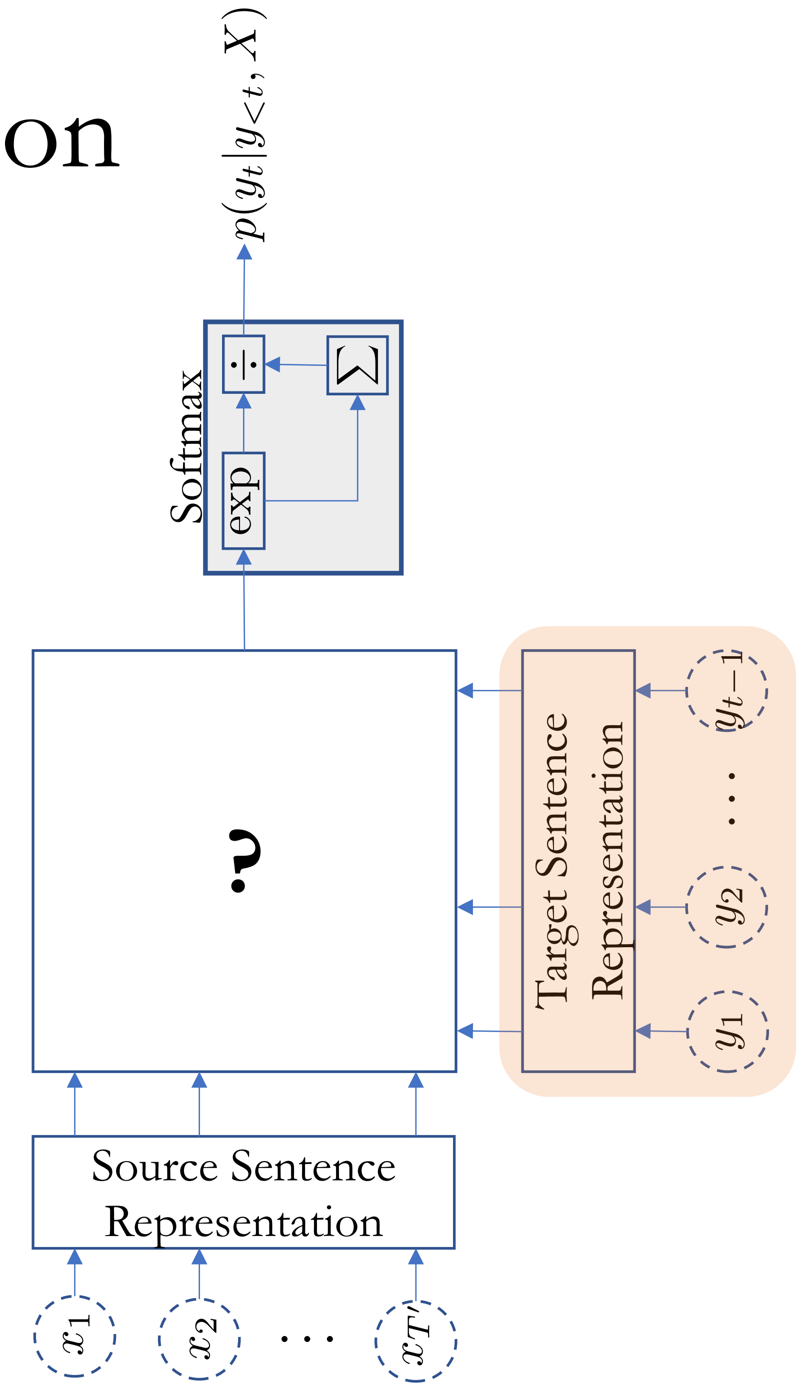
2. Target prefix representation
    - A unidirectional recurrent network



    - Compression of the target prefix

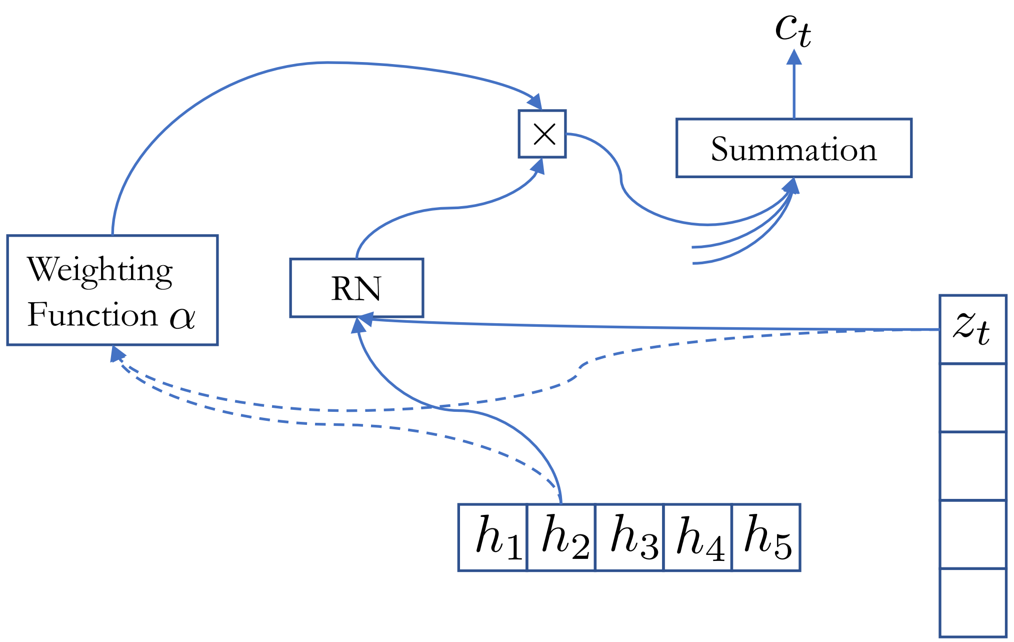
$$z_t = \mathrm{RNN}_{\mathrm{decoder}}(z_{t-1}, y_{t-1})$$

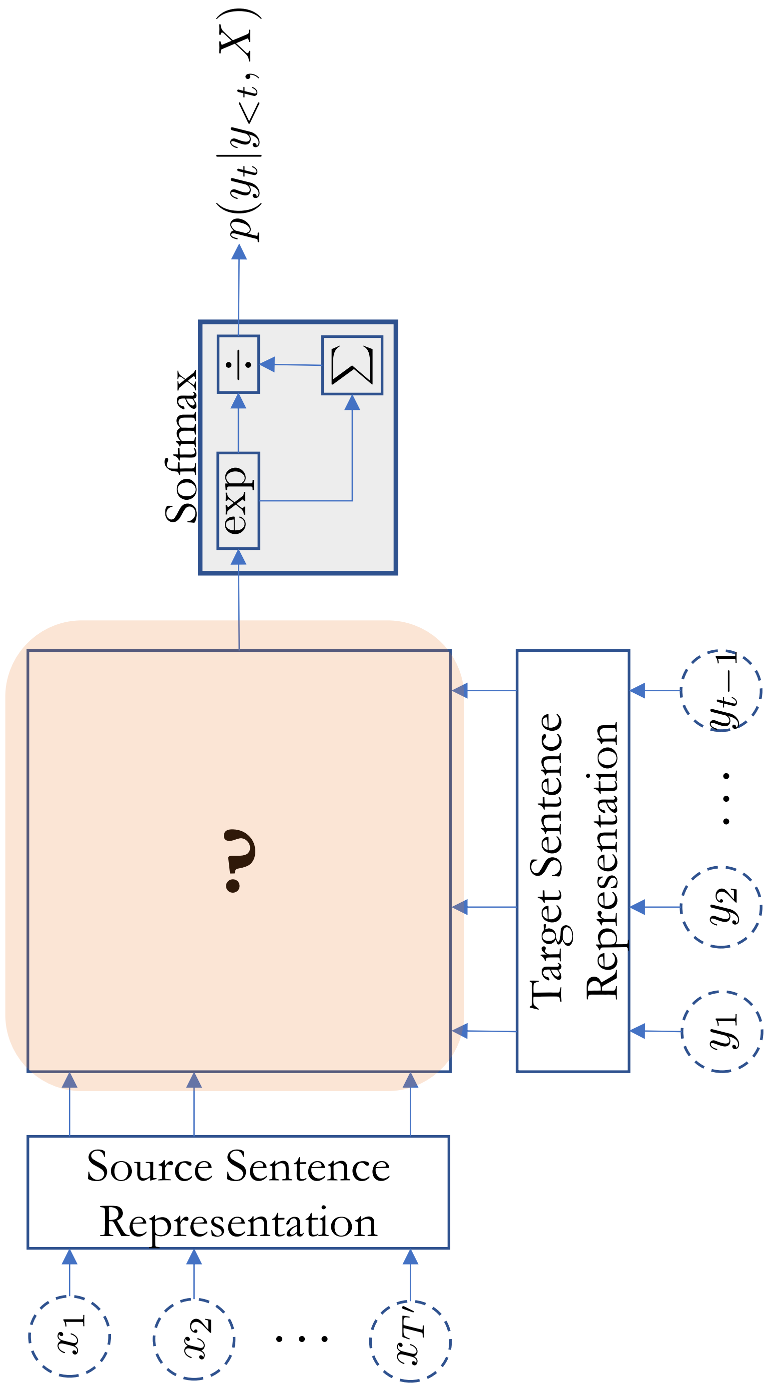
    - Summarizes what has been translated so far



RNN Neural Machine Translation
[Bahdanau et al., 2015]

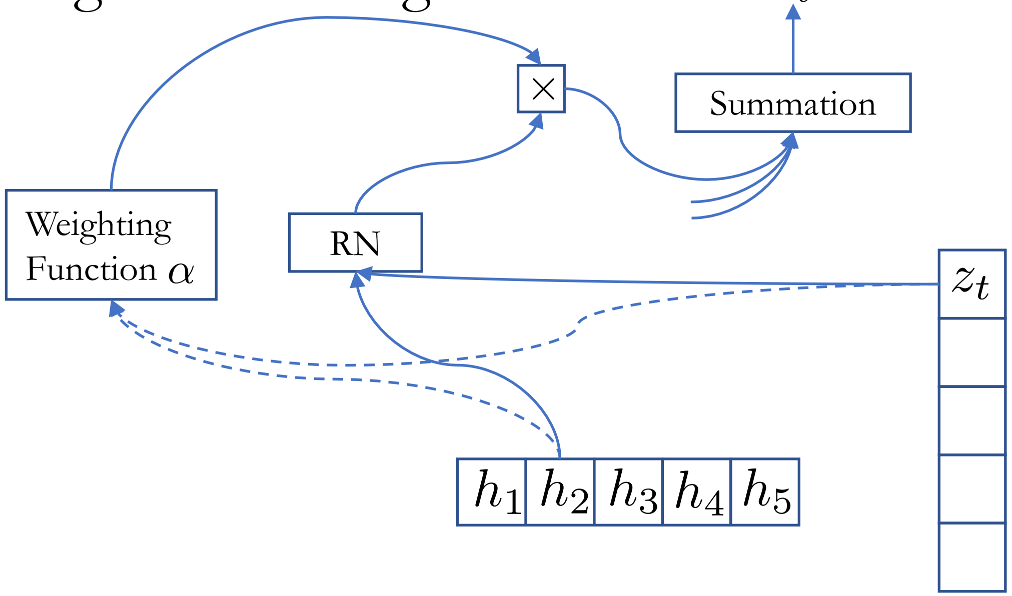
3. Attention mechanism
   - Which part of the source sentence is relevant for predicting the next target token?
   - Recall self-attention from Lecture 2



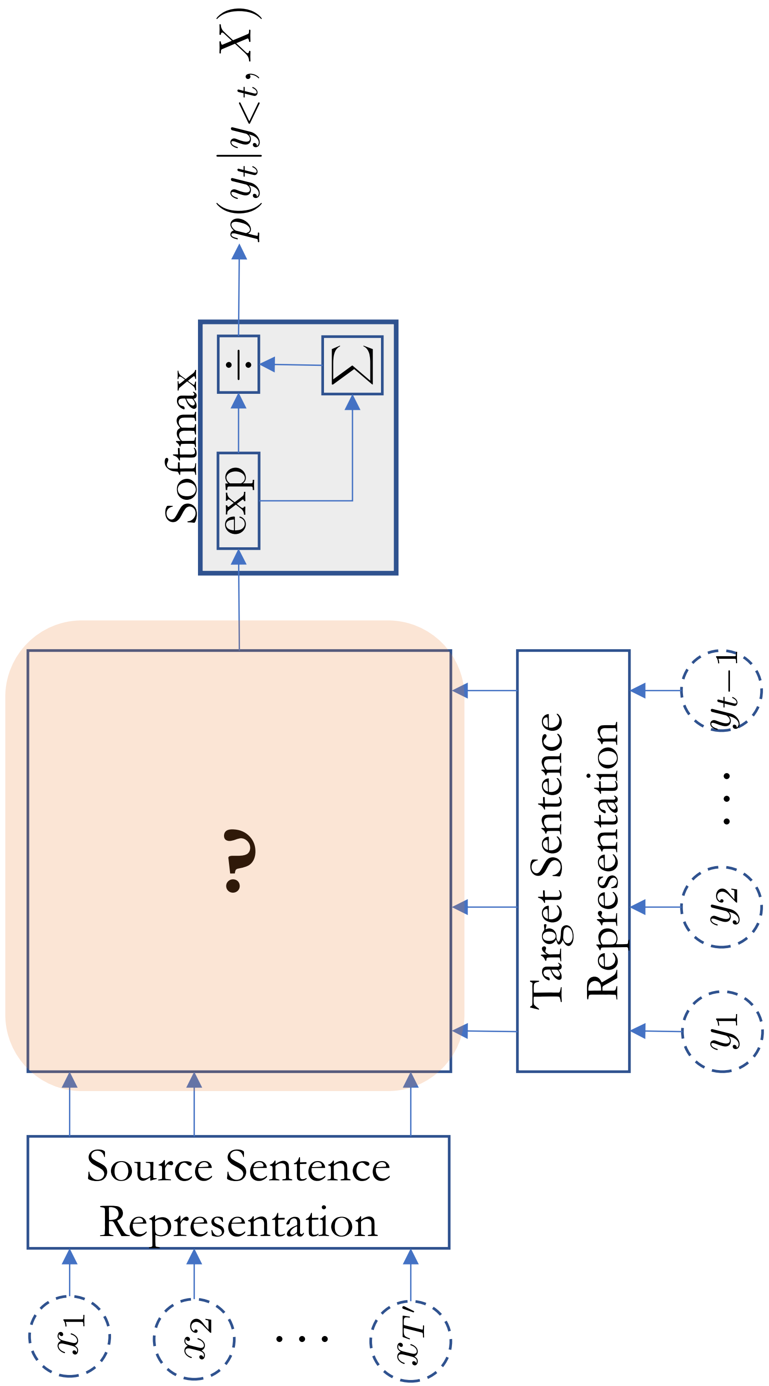


RNN Neural Machine Translation
[Bahdanau et al., 2015]

3. Attention mechanism
   - Which part of the source sentence is relevant for predicting the next target token?

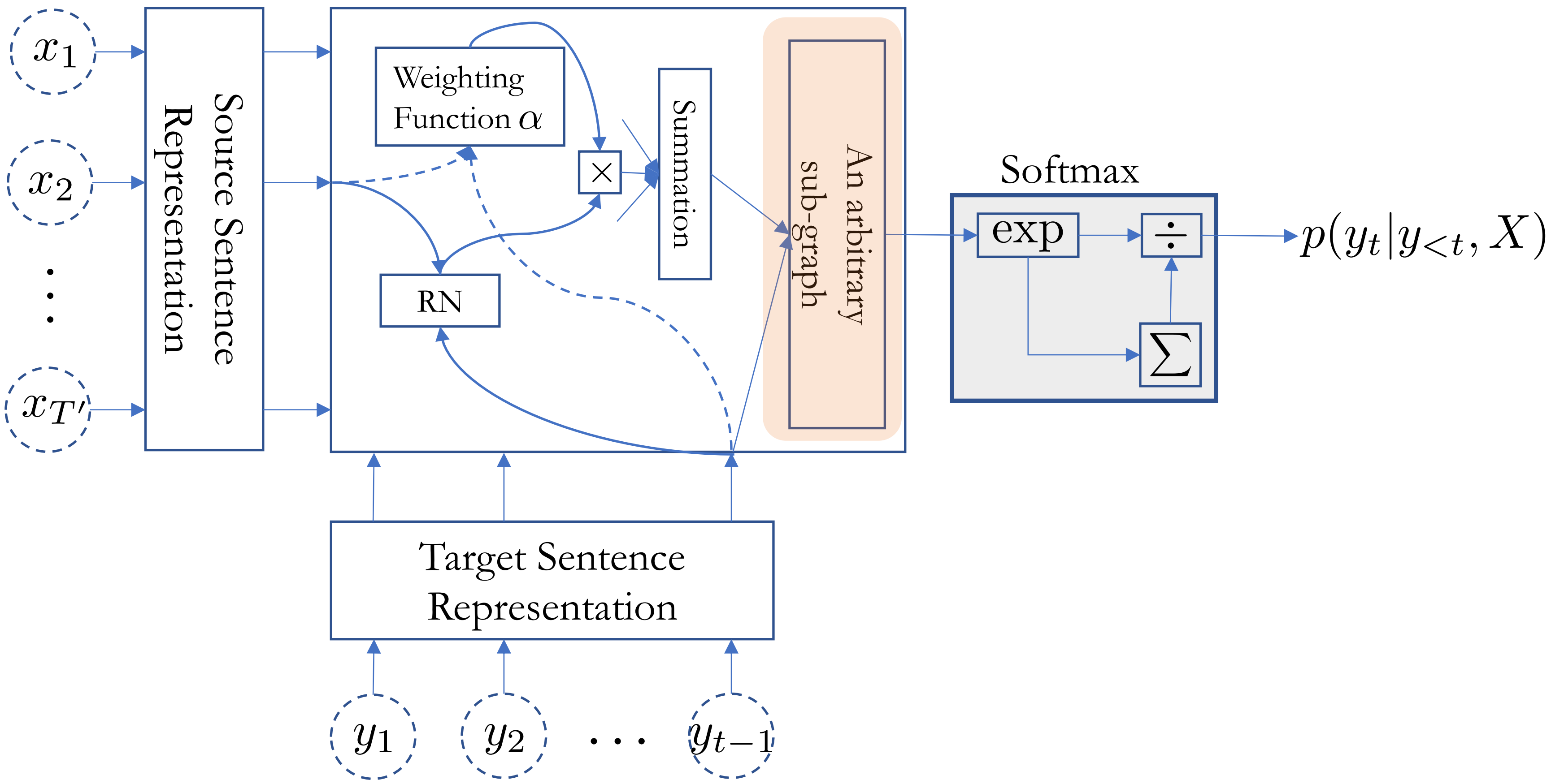


   - Time-dependent source context vector c_t



RNN Neural Machine Translation
[Bahdanau et al., 2015]

4. Fuse the source context vector and target prefix vector
   - Combines z_t and c_t into a single vector

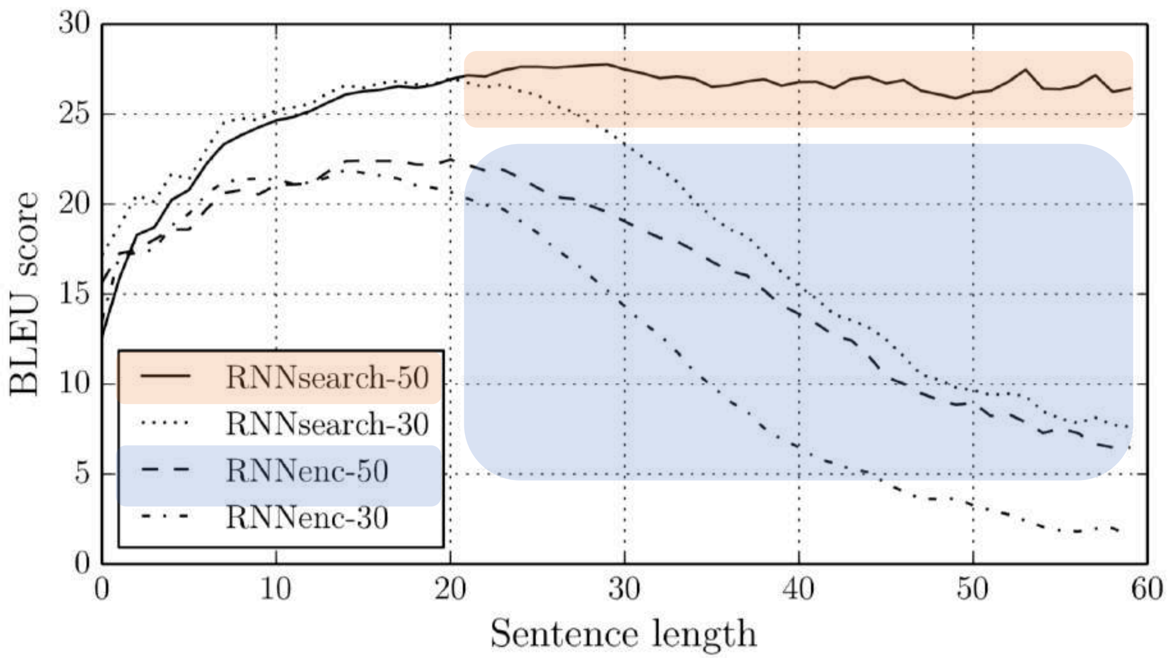


RNN Neural Machine Translation

- Conceptual process
  1. Encode: read the entire source sentence to know what to translate
  2. Attention: at each step, decide which source token(s) to translate next
  3. Decode: based on what has been translated and what need to be translated, predict the next target token.
  4. Repeat 2-3 until the <end-of-sentence> special token is generated.

RNN Neural Machine Translation

- The model is not pressured to compress the entire source sentence into a single, fixed-size vector:
  - Greatly improves the translation quality, especially of long sentences.
  - Much more efficient: less parameters are necessary.
- Bahdanau et al. [2015] showed for the first time the machine translation purely based on neural networks could be as good as then-state-of-the-art alternatives (e.g., PBMT).



RNN Neural Machine Translation

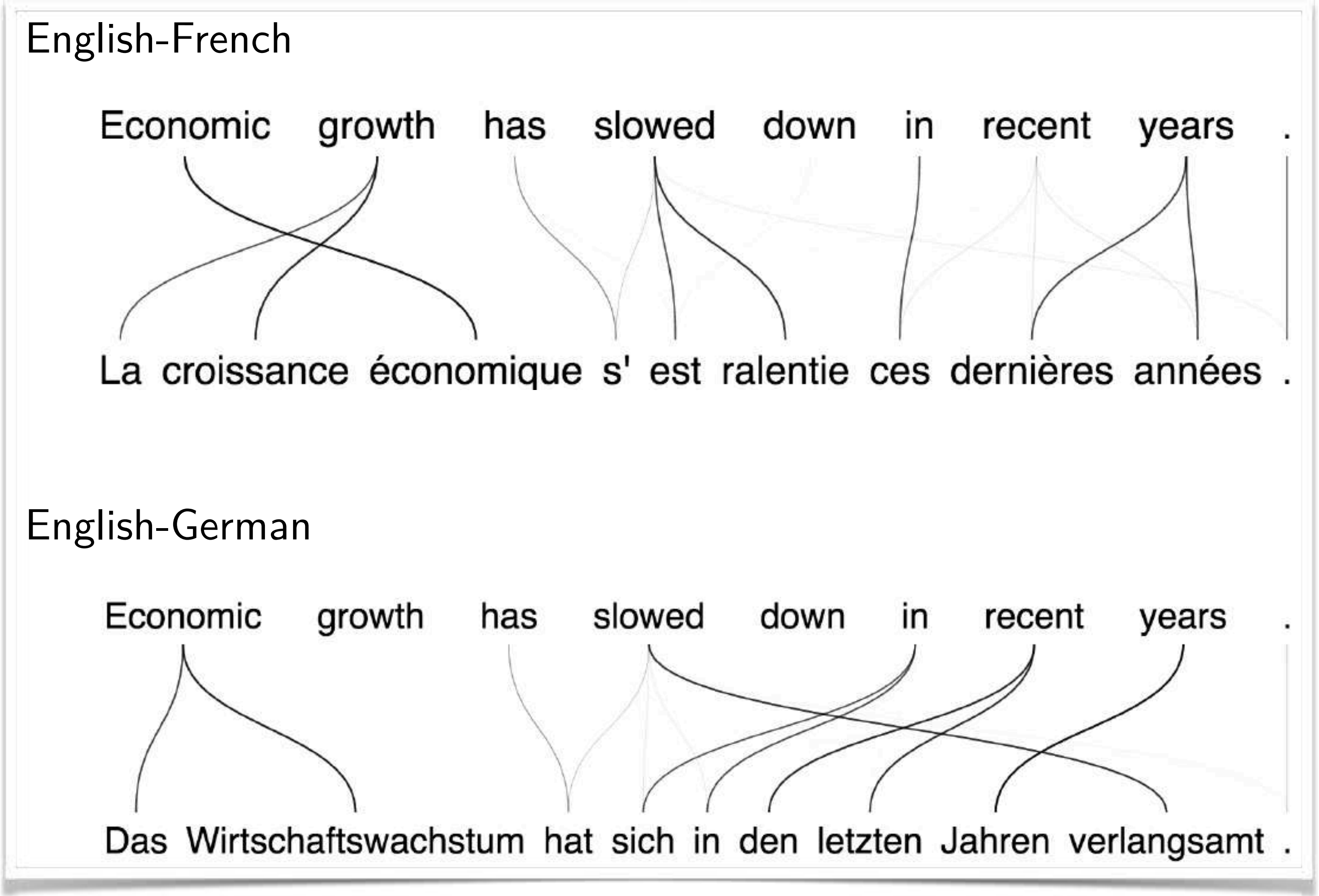
- **Source**: *An admitting privilege is the right of a doctor to admit a patient to a hospital or a medical centre to carry out a diagnosis or a procedure, based on his status as a health care worker at a hospital.*
- **When collapsed:** *Un privilège d'admission est le droit d'un médecin de reconnaître un patient à l'hôpital ou un centre médical d'un diagnostic ou de prendre un diagnostic en fonction de son état de santé.*
- **RNNSearch**: *Un privilège d'admission est le droit d'un médecin d'admettre un patient à un hôpital ou un centre médical pour effectuer un diagnostic ou une procédure, selon son statut de travailleur des soins de santé à l'hôpital.*

RNN Neural Machine Translation

- **Source**: *An admitting privilege is the right of a doctor to admit a patient to a hospital or a medical centre* <u>*to carry out a diagnosis or a procedure, based on his status as a health care worker at a hospital.*</u>
- **When collapsed:** *Un privilège d'admission est le droit d'un médecin de reconnaître un patient à l'hôpital ou un centre médical* <u>*d'un diagnostic ou de prendre un diagnostic en fonction de son état de santé.*</u>
- **RNNSearch**: *Un privilège d'admission est le droit d'un médecin d'admettre un patient à un hôpital ou un centre médical* <u>*pour effectuer un diagnostic ou une procédure, selon son statut de travailleur des soins de santé à l'hôpital.*</u>

RNN Neural Machine Translation

- Sensible alignment between source and target tokens
- Capture long-range reordering/dependencies
- Without strong supervision on the alignment
  - Weakly supervised learning



(Bahdanau et al., 2015; Jean et al., 2015)

In practice,

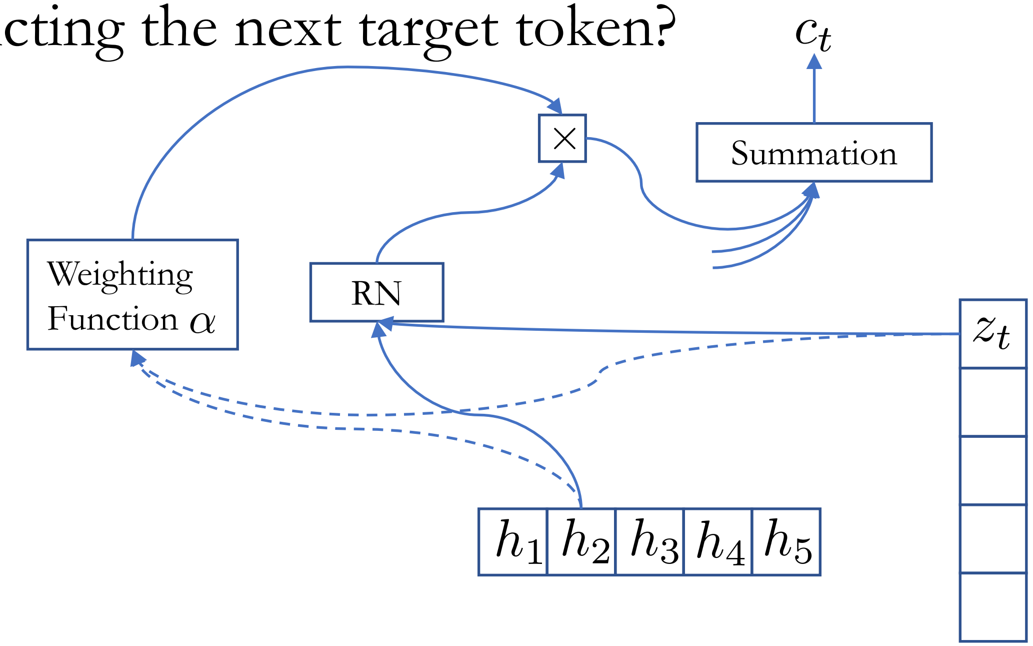
- Many excellent open-source packages exist:
  - Marian-NMT https://marian-nmt.github.io/
    - Compute backend: C++
    - Maximal efficiency
    - Supported by Microsoft Translate
  - FairSeq https://github.com/facebookresearch/fairseq
    - Compute backend: PyTorch
    - Supported by Facebook AI Research
  - Tensor2Tensor https://github.com/tensorflow/tensor2tensor
    - Compute backend: TensorFlow
    - Supported by Google

Attention Mechanism

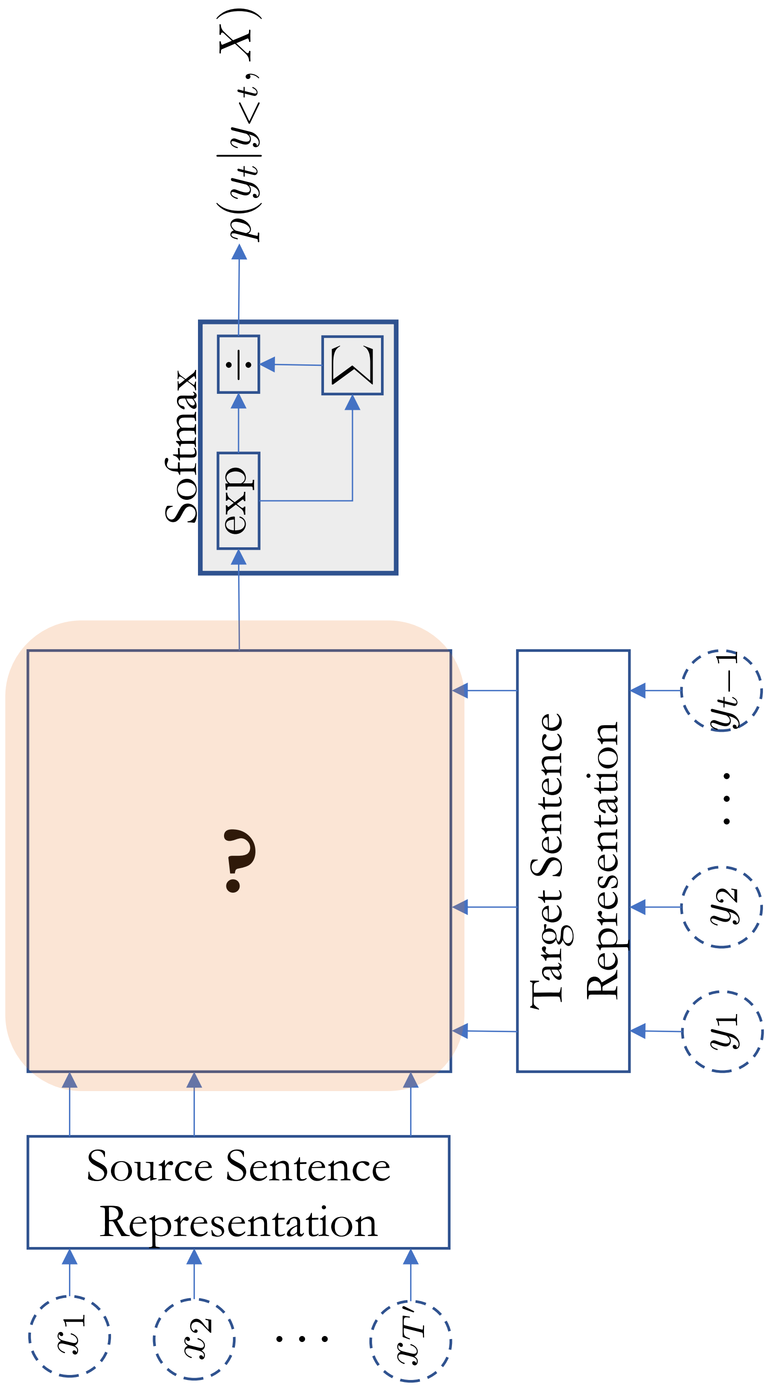
Delving deeper into the attention mechanism

RNN Neural Machine Translation
[Bahdanau et al., 2015]

3. Attention mechanism
   - Which part of the source sentence is relevant for predicting the next target token?

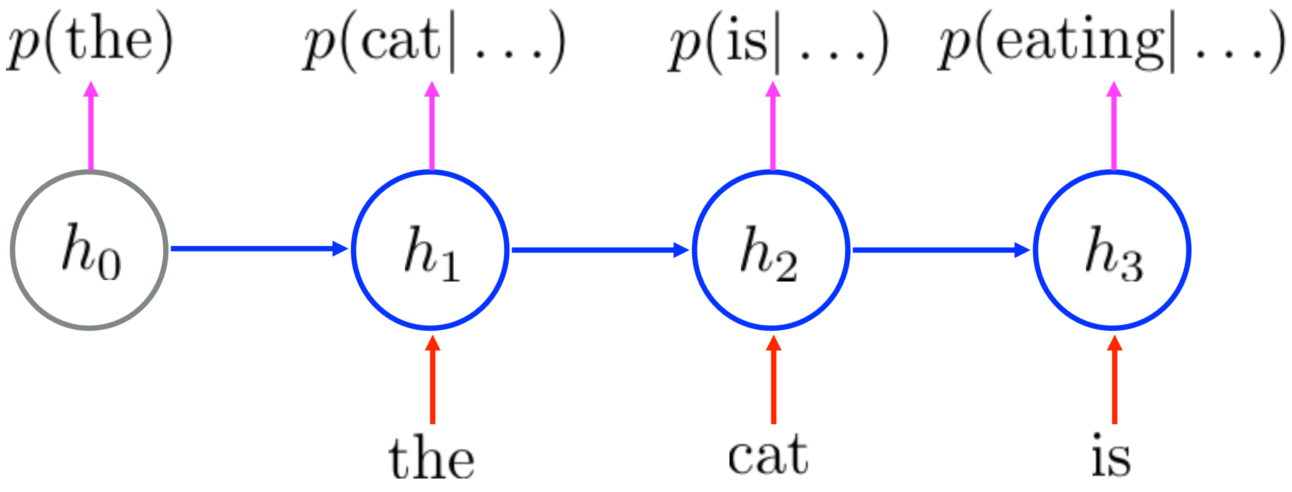


   - Time-dependent source context vector c_t



Rewind: Recurrent Language Model

Example) $p(\text{the}, \text{cat}, \text{is}, \text{eating})$



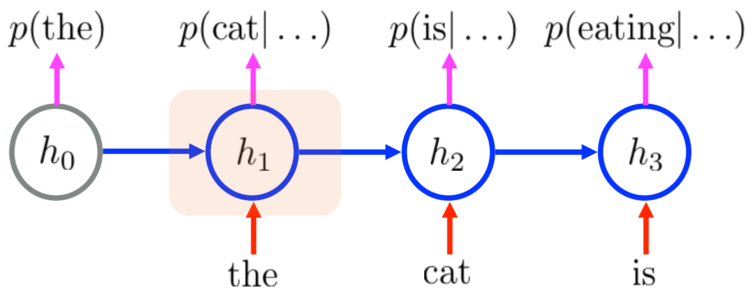
Read, *Update and Predict*

Building a Recurrent Language Model

Transition Function $h_t = f(h_{t-1}, x_{t-1})$

- Inputs
  1. Previous word $x_{t-1} \in \{1, 2, \ldots, |V|\}$
  2. Previous state $h_{t-1} \in \mathbb{R}^d$
- Parameters
  1. Input weight matrix $W \in \mathbb{R}^{|V| \times d}$
  2. Transition weight matrix $U \in \mathbb{R}^{d \times d}$
  3. Bias vector $b \in \mathbb{R}^d$
- Naïve Transition Function

$$f(h_{t-1}, x_{t-1}) = \tanh(W\left[x_{t-1}\right] + Uh_{t-1} + b)$$



Building a Recurrent Language Model

Transition Function $$h_t = f(h_{t-1}, x_{t-1})$$

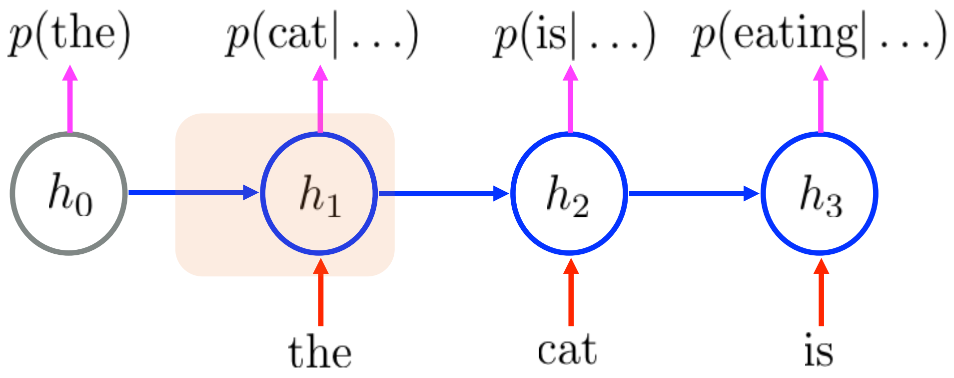
- Naïve Transition Function

$$f(h_{t-1}, x_{t-1}) = \tanh(W\,[x_{t-1}] + U h_{t-1} + b)$$

Element-wise nonlinear transformation

Linear transformation of previous state

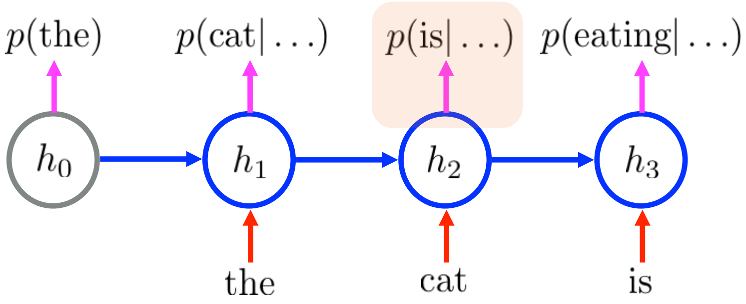
Trainable word vector



Building a Recurrent Language Model

Readout Function $p(x_t = w | x_{<t}) = g_w(h_t)$

- Inputs
  1. Current state $h_t \in \mathbb{R}^d$
- Parameters
  1. Readout weight matrix $R \in \mathbb{R}^{|V| \times d}$
  2. Bias vector $c \in \mathbb{R}^{|V|}$



- Softmax Readout

$$p(x_t = w | x_{<t}) = g_w(h_t) = \frac{\exp(R[w]^\top h_t + c_w)}{\sum_{i=1}^{|V|} \exp(R[i]^\top h_t + c_i)}$$

Building a Recurrent Language Model

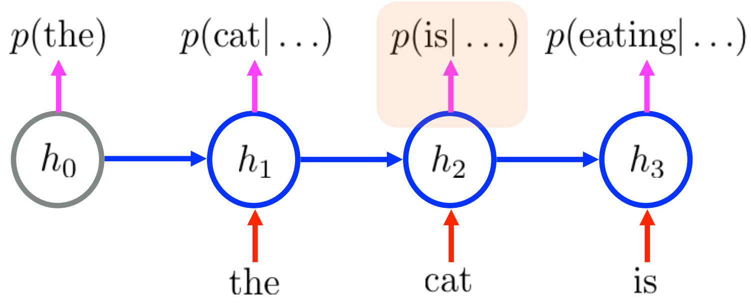
Readout Function $p(x_t = w | x_{<t}) = g_w(h_t)$

$$p(x_t = w | x_{<t}) = g_w(h_t) = \frac{\exp\left(R\left[w\right]^\top h_t + c_w\right)}{\sum_{i=1}^{|V|} \exp\left(R\left[i\right]^\top h_t + c_i\right)}$$

Exponentiation

Normalization

Compatibility between a trainable word vector and the hidden state



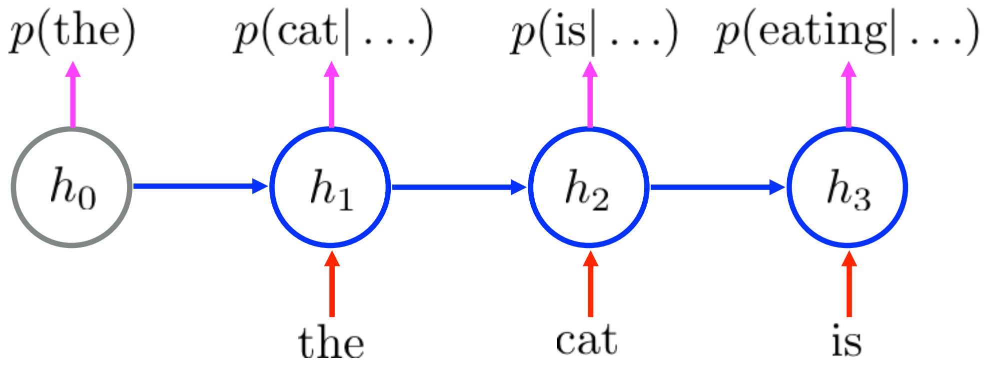
Training a Recurrent Language Model

- Log-probability of one training sentence

$$\log p(x_1^n, x_2^n, \ldots, x_{T^n}^n) = \sum_{t=1}^{T^n} \log p(x_t^n | x_1^n, \ldots, x_{t-1}^n)$$

- Training set $D = \{X^1, X^2, \ldots, X^N\}$
- Log-likelihood Functional

$$\mathcal{L}(\theta, D) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T^n} \log p(x_t^n | x_1^n, \ldots, x_{t-1}^n)$$



Minimize $-\mathcal{L}(\theta, D)$ *!!*

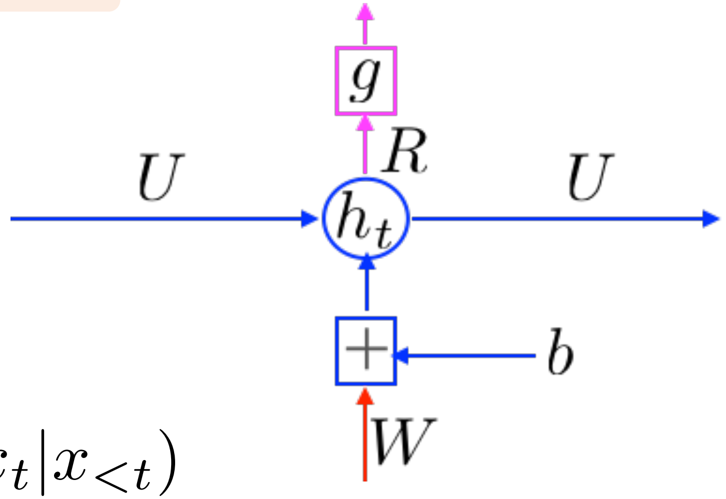
Backpropagation through Time

How do we compute $\nabla \mathcal{L}(\theta, X)$ *?*

- Decompose the per-sample cost into per-step cost functions

$$\nabla \mathcal{L}(\theta, X) = \sum_{t=1}^{T} \nabla \log p(x_t | x_{<t}, \theta)$$

- Compute per-step cost function from time $t = T$
  1. Cost derivative $\partial \log p(x_t | x_{<t}) / \partial g$
  2. Gradient w.r.t. R: $\times \partial g / \partial R$
  3. Gradient w.r.t. h_t: $\times \partial g / \partial h_t + \partial h_{t+1} / \partial h_t \log p(x_t | x_{<t})$
  4. Gradient w.r.t. U: $\times \partial h_t / \partial U$
  5. Gradient w.r.t. b and W : $\times \partial h_t / \partial b$ and $\times \partial h_t / \partial W$
  6. Accumulate the gradient and $t \leftarrow t - 1$



Note: *I'm abusing math a lot here!!*

Backpropagation through Time

Intuitively, what's happening here?

1. Measure the influence of the past on the future

$$\frac{\partial \log p(x_{t+n}|x_{<t+n})}{\partial h_t} = \frac{\partial \log p(x_{t+n}|x_{<t+n})}{\partial g} \frac{\partial g}{\partial h_{t+n}} \frac{\partial h_{t+n}}{\partial h_{t+n-1}} \cdots \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial h_t}{\partial x_t}$$

2. If I perturb the input at t, how does it affect $p(x_{t+n}|x_{<t+n})$?

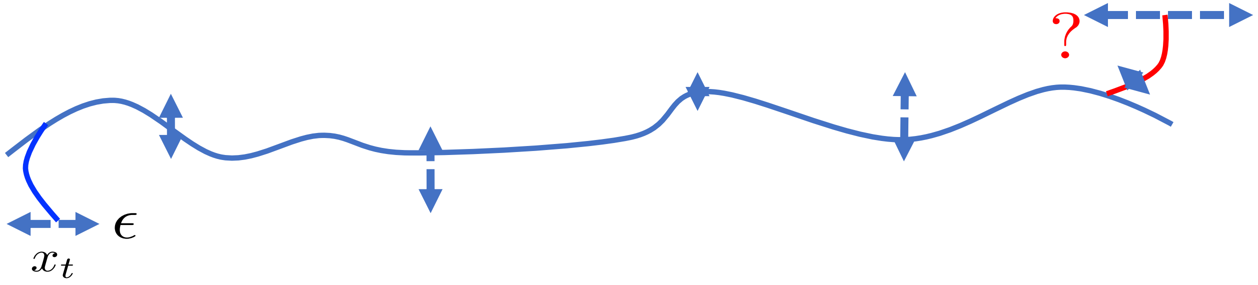
Backpropagation through Time

Intuitively, what's happening here?

1. Measure the influence of the past on the future

$$\frac{\partial \log p(x_{t+n}|x_{<t+n})}{\partial h_t} = \frac{\partial \log p(x_{t+n}|x_{<t+n})}{\partial g} \frac{\partial g}{\partial h_{t+n}} \frac{\partial h_{t+n}}{\partial h_{t+n-1}} \cdots \frac{\partial h_{t+1}}{\partial h_t} \frac{\partial h_t}{\partial x_t}$$

2. If I perturb the input at t , how does it affect $p(x_{t+n}|x_{<t+n})$?



3. Change the parameters θ so as to maximize $p(x_{t+n}|x_{<t+n})$

Backpropagation through Time

Intuitively, what's happening here?

1. Measure the influence of the past on the future

$$\frac{\partial \log p(x_{t+n}|x_{<t+n})}{\partial h_t} = \frac{\partial \log p(x_{t+n}|x_{<t+n})}{\partial g} \frac{\partial g}{\partial h_{t+n}} \frac{\partial h_{t+n}}{\partial h_{t+n-1}} \cdots \frac{\partial h_{t+1}}{\partial h_t}$$

2. With a naïve transition function

$$f(h_{t-1}, x_{t-1}) = \tanh(W\,[x_{t-1}] + U h_{t-1} + b)$$

We get $\displaystyle \frac{\partial J_{t+n}}{\partial h_t} = \frac{\partial J_{t+n}}{\partial g} \frac{\partial g}{\partial h_{t+N}} \underbrace{\prod_{n=1}^{N} U^\top \mathrm{diag}\left(\frac{\partial \tanh(a_{t+n})}{\partial a_{t+n}}\right)}_{\textit{Problematic!}}$

– Bengio et al. (1994)

Backpropagation through Time

Gradient either ***vanishes*** *or* ***explodes*** Pascanu et al. (2013)

- What happens? $$\frac{\partial J_{t+n}}{\partial h_t} = \frac{\partial J_{t+n}}{\partial g} \frac{\partial g}{\partial h_{t+N}} \underbrace{\prod_{n=1}^{N} U^\top \mathrm{diag}\left(\frac{\partial \tanh(a_{t+n})}{\partial a_{t+n}}\right)}$$

1. The gradient *likely* explodes if

$$e_{\max} \geq \frac{1}{\max \tanh'(x)} = 1$$

2. The gradient *likely* vanishes if

$$e_{\max} < \frac{1}{\max \tanh'(x)} = 1$$

$e_{\max}$: largest eigenvalue of U

Backpropagation through Time

Let the (norm of the) gradient explode!

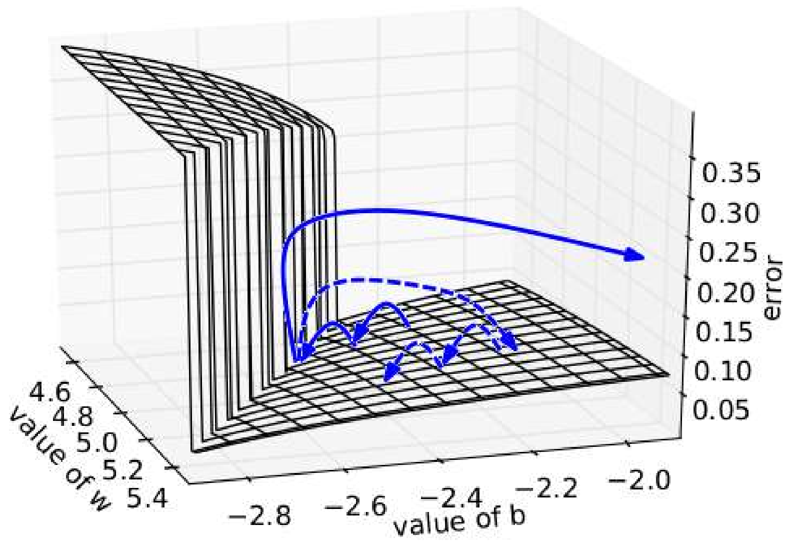
- *"when gradients explode so does the curvature along v, leading to a wall in the error surface"*

- Simple solution: Gradient Clipping
  1. Norm clipping

$$\tilde{\nabla} \leftarrow \begin{cases} \frac{c}{\|\nabla\|}\nabla & \text{,if } \|\nabla\| \geq c \\ \nabla & \text{,otherwise} \end{cases}$$

  2. Element-wise clipping

$$\nabla_i \leftarrow \min(c, \nabla_i), \text{ for all } i \in \{1, \ldots, \dim\nabla\}$$



Pascanu et al. (2013)

Backpropagation through Time

Vanishing gradient is super-problematic

- We cannot tell whether
  1. no long-term dependency between t and $t+n$ in data, or
  2. wrong configuration of parameters:

$$e_{\max}(U) < \frac{1}{\max \tanh'(x)}$$

- We only observe $\left\| \frac{\partial h_{t+N}}{\partial h_t} \right\| = \left\| \prod_{n=1}^{N} U^\top \mathrm{diag}\left(\frac{\partial \tanh(a_{t+n})}{\partial a_{t+n}} \right) \right\| \to 0$

Backpropagation through Time

Vanishing gradient is super-problematic

- Let's just say there is such a long-term dependency. Then,
  - "*we … force the network to increase the norm of* $\frac{\partial h_{t+N}}{\partial h_t}$ *at the expense of larger errors*"

Pascanu et al. (2013)

- This can be done by regularizing

$$\sum_{t=1}^{T}\left(1-\frac{\left\|\frac{\partial\tilde{C}}{\partial\mathbf{h}_{t+1}}\frac{\partial\mathbf{h}_{t+1}}{\partial\mathbf{h}_t}\right\|}{\left\|\frac{\partial\tilde{C}}{\partial\mathbf{h}_{t+1}}\right\|}\right)^2$$

- This doesn't seem like a great nor easy way to deal with the vanishing gradient.

Gated Recurrent Unit

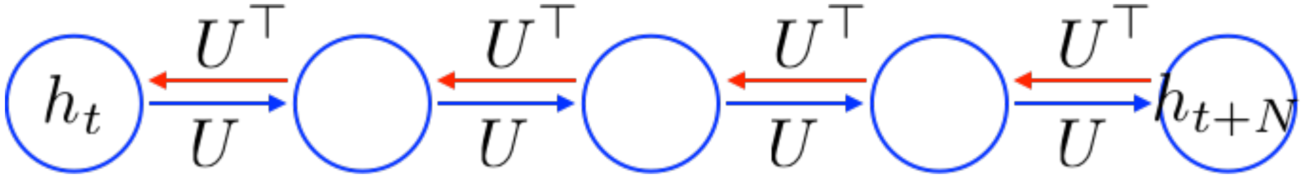
- Perhaps, the problem is with the naïve transition function

$$f(h_{t-1}, x_{t-1}) = \tanh(W\left[x_{t-1}\right] + Uh_{t-1} + b)$$

- With it, the temporal derivative is

$$\frac{\partial h_{t+1}}{\partial h_t} = U^\top \frac{\partial \tanh(a)}{\partial a}$$

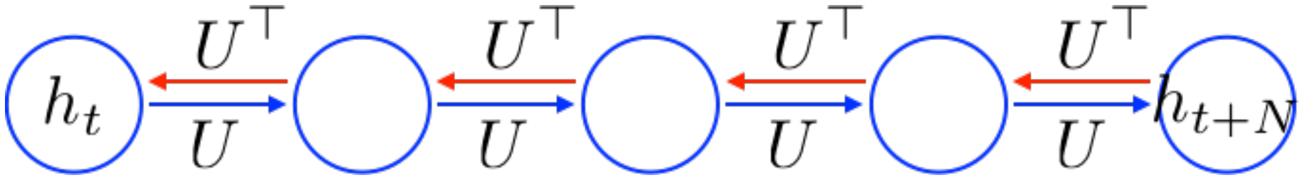
- It implies that the error must backpropagate through all the intermediate nodes:



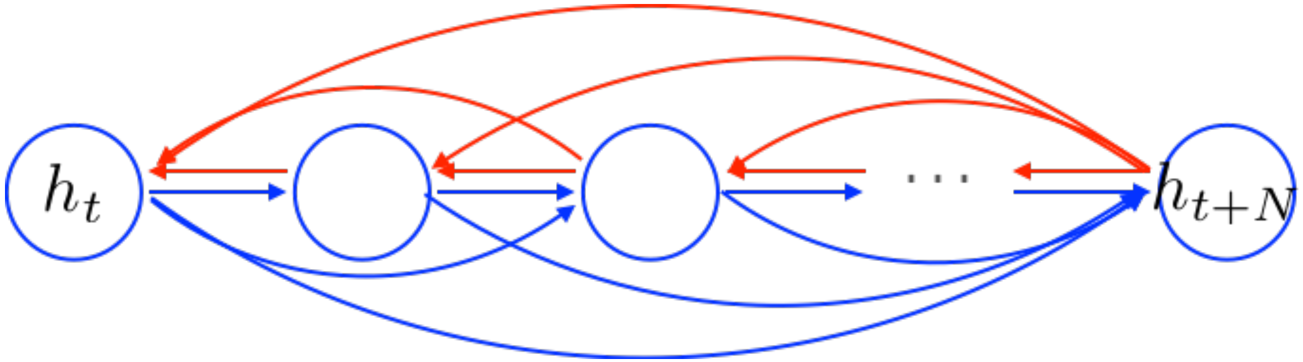
Gated Recurrent Unit

$$f(h_{t-1}, x_{t-1}) = \tanh(W[x_{t-1}] + Uh_{t-1} + b)$$

- It implies that the error must backpropagate through all the intermediate nodes:



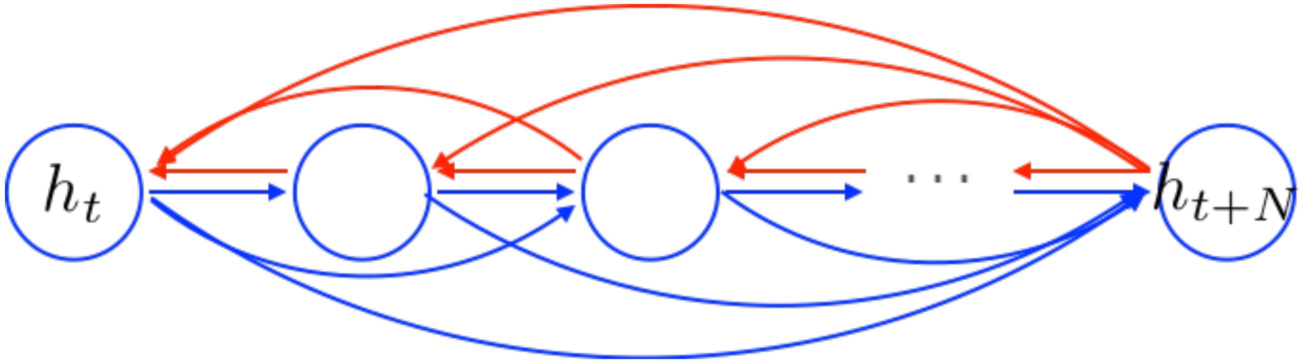
- Perhaps we can create shortcut connections.



Gated Recurrent Unit

$$f(h_{t-1}, x_{t-1}) = \tanh(W[x_{t-1}] + Uh_{t-1} + b)$$

- Perhaps we can create *adaptive* shortcut connections.



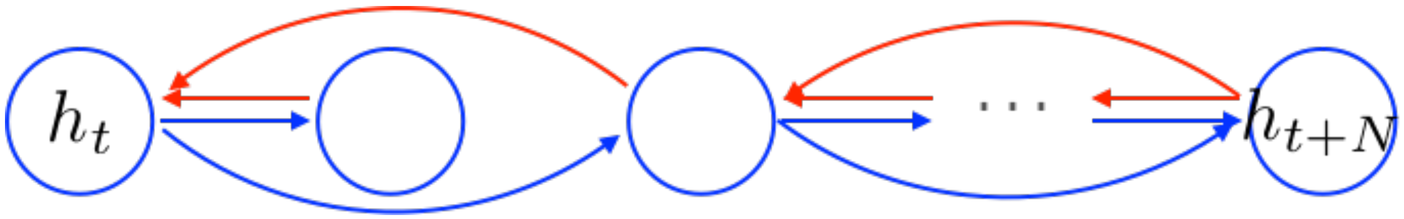
$$f(h_{t-1}, x_{t-1}) = u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1}$$

- Candidate Update $\tilde{h}_t = \tanh(W[x_{t-1}] + Uh_{t-1} + b)$
- Update gate $u_t = \sigma(W_u[x_{t-1}] + U_u h_{t-1} + b_u)$

Gated Recurrent Unit

$$f(h_{t-1}, x_{t-1}) = \tanh(W[x_{t-1}] + Uh_{t-1} + b)$$

- We also let the network prune unnecessary shortcuts *adaptively*.

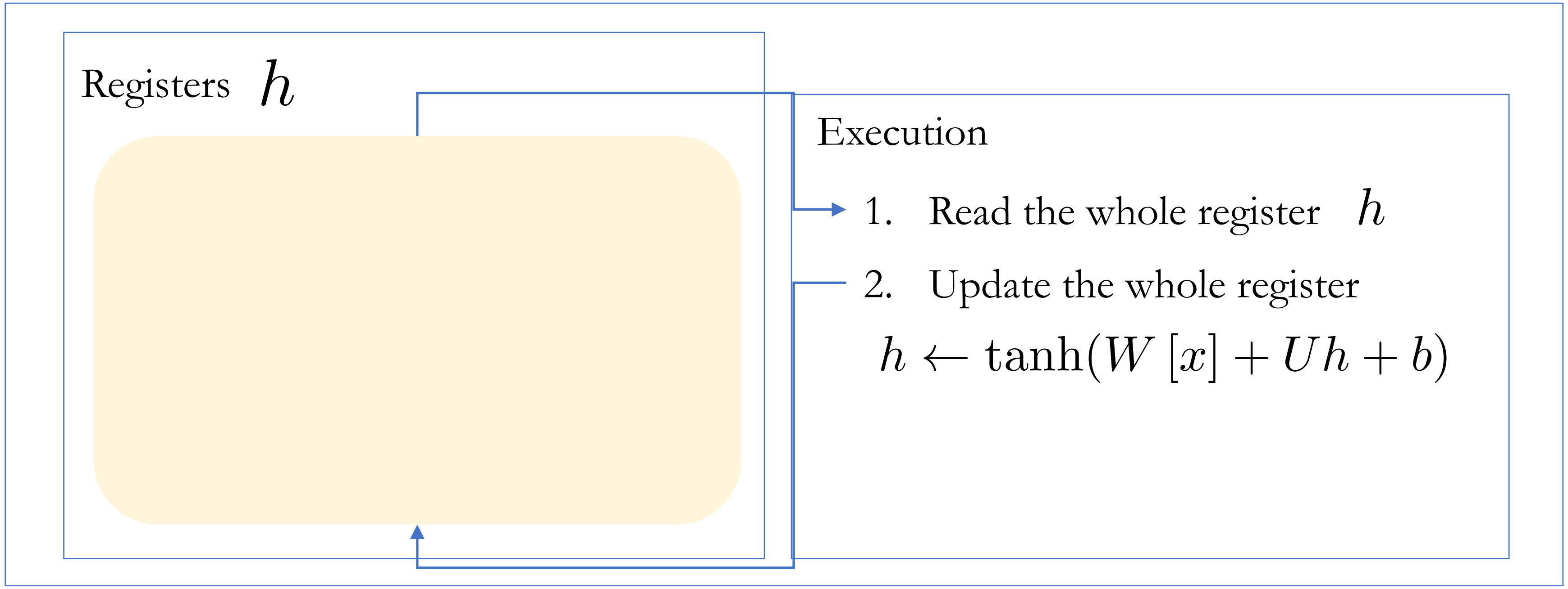


$$f(h_{t-1}, x_{t-1}) = u_t \odot \tilde{h}_t + (1 - u_t) \odot h_{t-1}$$

- Candidate Update $\tilde{h}_t = \tanh(Wx_{t-1} + U(r_t \odot h_{t-1}) + b)$
- Reset gate $r_t = \sigma(W_r x_{t-1} + U_r h_{t-1} + b_r)$
- Update gate $u_t = \sigma(W_u[x_{t-1}] + U_u h_{t-1} + b_u)$

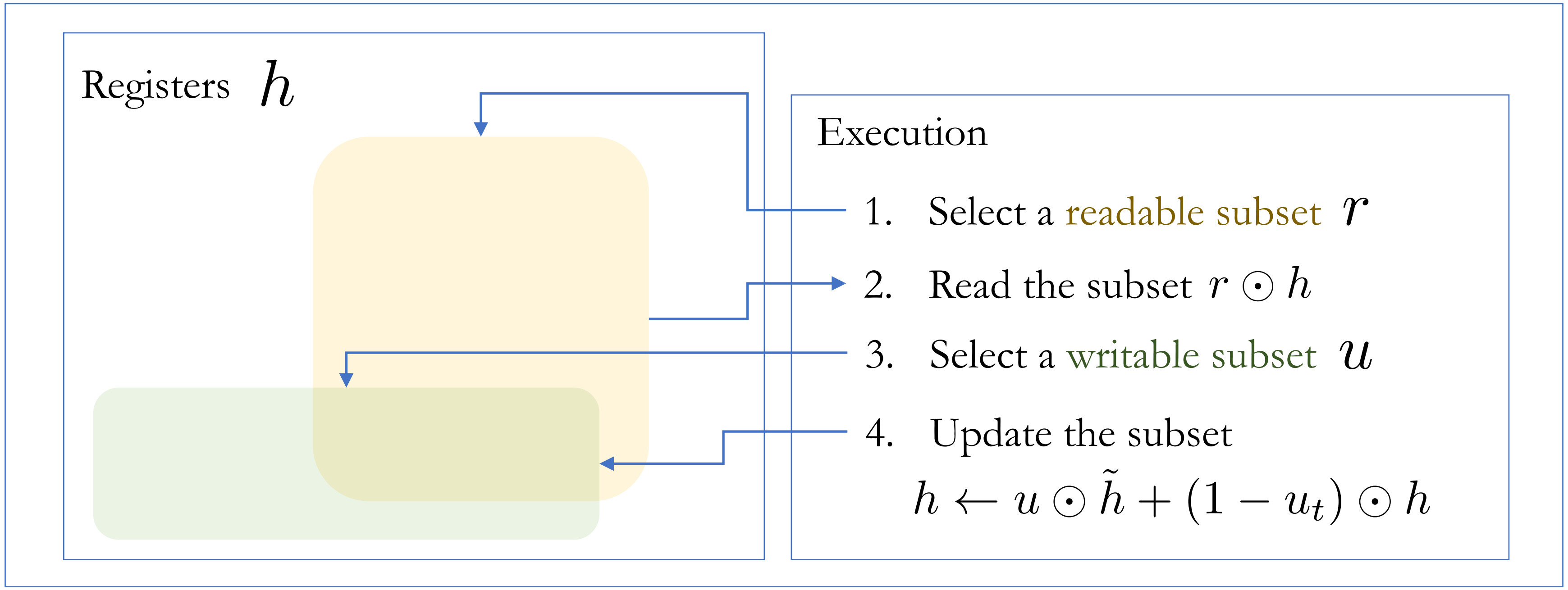
Gated Recurrent Unit

tanh-RNN vs CPU



Gated Recurrent Unit

GRU vs CPU



Clearly gated recurrent units* are much more realistic.

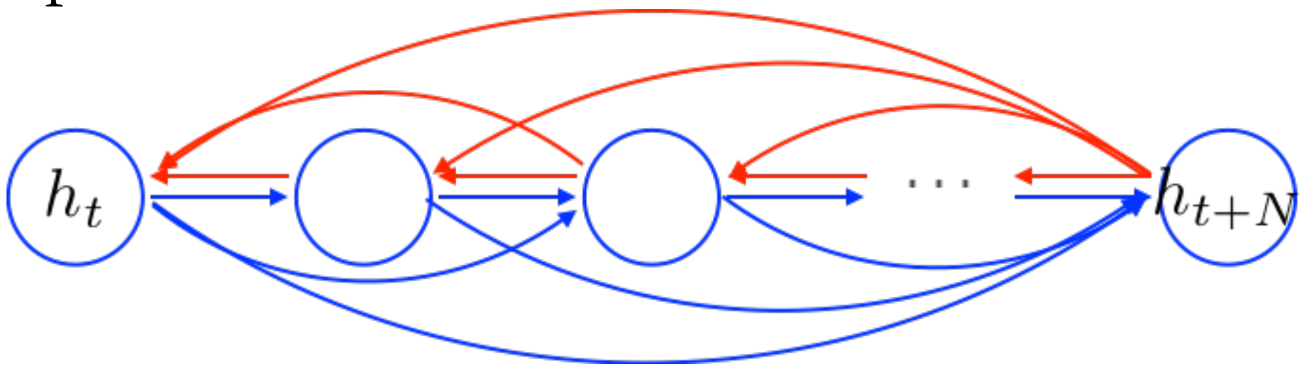
** By gated recurrent units, I refer to both LSTM and GRU.*

Gated recurrent units to attention

- A key idea behind LSTM and GRU is the additive update

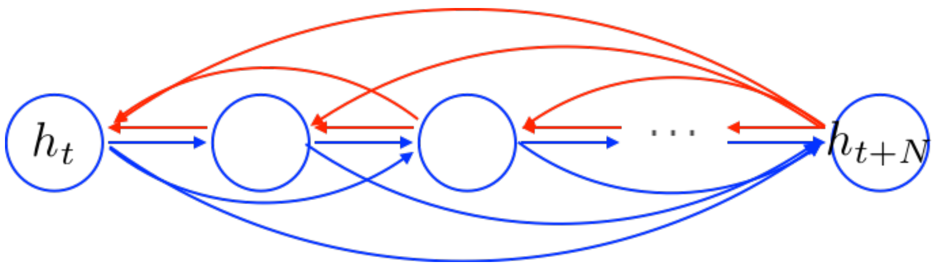
$$h_t = u_t \odot h_{t-1} + (1 - u_t) \odot \tilde{h}_t, \text{ where } \tilde{h}_t = f(x_t, h_{t-1})$$

- This additive update creates linear short-cut connections



Side-note: gated recurrent units to attention

- What are these shortcuts?



- When unrolled, it's a weighted combination of all previous hidden vectors:

$$
\begin{aligned}
h_t =& u_t \odot h_{t-1} + (1-u_t) \odot \tilde{h}_t, \\
=& u_t \odot (u_{t-1} \odot h_{t-2} + (1-u_{t-1}) \odot \tilde{h}_{t-1}) + (1-u_t) \odot \tilde{h}_t, \\
=& u_t \odot (u_{t-1} \odot (u_{t-2} \odot h_{t-3} + (1-u_{t-2}) \odot \tilde{h}_{t-2}) + (1-u_{t-1}) \odot \tilde{h}_{t-1}) + (1-u_t) \odot \tilde{h}_t, \\
& \vdots \\
=& \sum_{i=1}^{t} \left(\prod_{j=i}^{t-i+1} u_j \right) \left(\prod_{k=1}^{i-1} (1-u_k) \right) \tilde{h}_i
\end{aligned}
$$

Gated recurrent units to causal attention

1. Can we "free" these dependent weights?
2. Can we "free" candidate vectors?
3. Can we separate keys and values?
4. Can we have multiple attention heads?

$$h_t = \sum_{i=1}^{t} \left(\prod_{j=i}^{t-i+1} u_j \right) \left(\prod_{k=1}^{i-1} (1 - u_k) \right) \tilde{h}_i \quad \mathbf{0}$$

$$h_t = \sum_{i=1}^{t} \alpha_i \tilde{h}_i, \text{ where } \alpha_i \propto \exp(\mathrm{ATT}(\tilde{h}_i, x_t)) \quad \mathbf{1}$$

$$h_t = \sum_{i=1}^{t} \alpha_i f(x_i), \text{ where } \alpha_i \propto \exp(\mathrm{ATT}(f(x_i), x_t)) \quad \mathbf{2}$$

$$h_t = \sum_{i=1}^{t} \alpha_i V(f(x_i)), \text{ where } \alpha_i \propto \exp(\mathrm{ATT}(K(f(x_i)), Q(x_t))) \quad \mathbf{3}$$

$$h_t = [h_t^1; \cdots; h_t^K], \text{ where } h_t^k = \sum_{i=1}^{t} \alpha_i^k V^k(f(x_i)), \text{ and } \alpha_i^k \propto \exp(\mathrm{ATT}(K^k(f(x_i)), Q^k(f(x_t)))) \quad \mathbf{4}$$

Gated recurrent units to non-causal attention

1. Look at the entire input sequence

$$h_t = [h_t^1; \cdots; h_t^K], \text{ where } h_t^k = \sum_{i=1}^{T} \alpha_i^k V^k(f(x_i)), \text{ and } \alpha_i^k \propto \exp(\text{ATT}(K^k(f(x_i)), Q^k(f(x_t))))$$

2. Give the sense of positions

$$h_t = [h_t^1; \cdots; h_t^K],$$

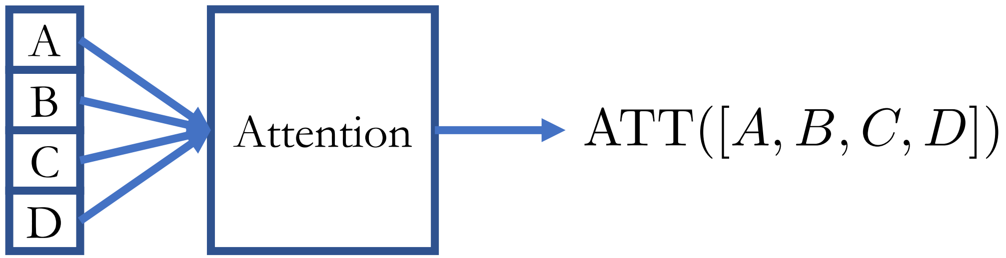
where

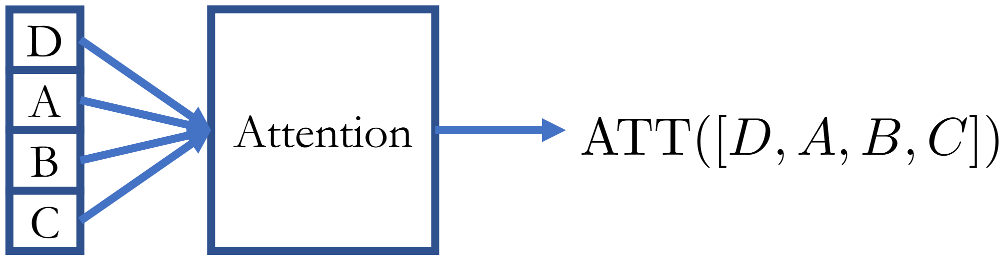
$$h_t^k = \sum_{i=1}^{T} \alpha_i^k V^k(f(x_i) + p(i)),$$

$$\alpha_i^k \propto \exp(\text{ATT}(K^k(f(x_i) + p(i)), Q^k(f(x_t) + p(i))))$$

Note on positional embedding

- Attention is position-invariant: $\mathrm{ATT}([A, B, C, D]) = \mathrm{ATT}([D, A, B, C])$





- Add position-specific vectors: positional embedding
  - Learned positional embedding [Sukhbataar et al., 2016]
  - Sinusoidal positional embedding [Vaswani et al., 2017]

Nonlinear Attention

- Attention is inherently linear, b/c it is a weighted sum of input vectors
  - f is often an identity function. p does not depend on the input.
  - V is often a linear transformation.

$$h_t^k = \sum_{i=1}^{T} \alpha_i^k V^k(f(x_i) + p(i))$$

- A post-attention nonlinear layer
  - g is a feedforward neural network and applied to each time step independently
  - For higher efficiency, g may apply to each head independently as well

$$h_t = g([h_t^1; \cdots ; h_t^K])$$

Full self-attention layer

$$h_t = g([h_t^1; \cdots ; h_t^K]),$$

where

$$h_t^k = \sum_{i=1}^{T} \alpha_i^k V^k(f(x_i) + p(i)),$$

$$\alpha_i^k \propto \exp(\mathrm{ATT}(K^k(f(x_i) + p(i)), Q^k(f(x_t) + p(i))))$$

Parametrization – Transformers

- Stack multiple layers of attention to build a transformer
  - Vaswani et al. [2017] – Attention is all you need
- A transformer block consists of
  1. Multi-headed attention
  2. Residual connection
  3. Feedforward layer
  4. Point-wise nonlinearity
  5. Residual connection
  6. (Layer) normalization

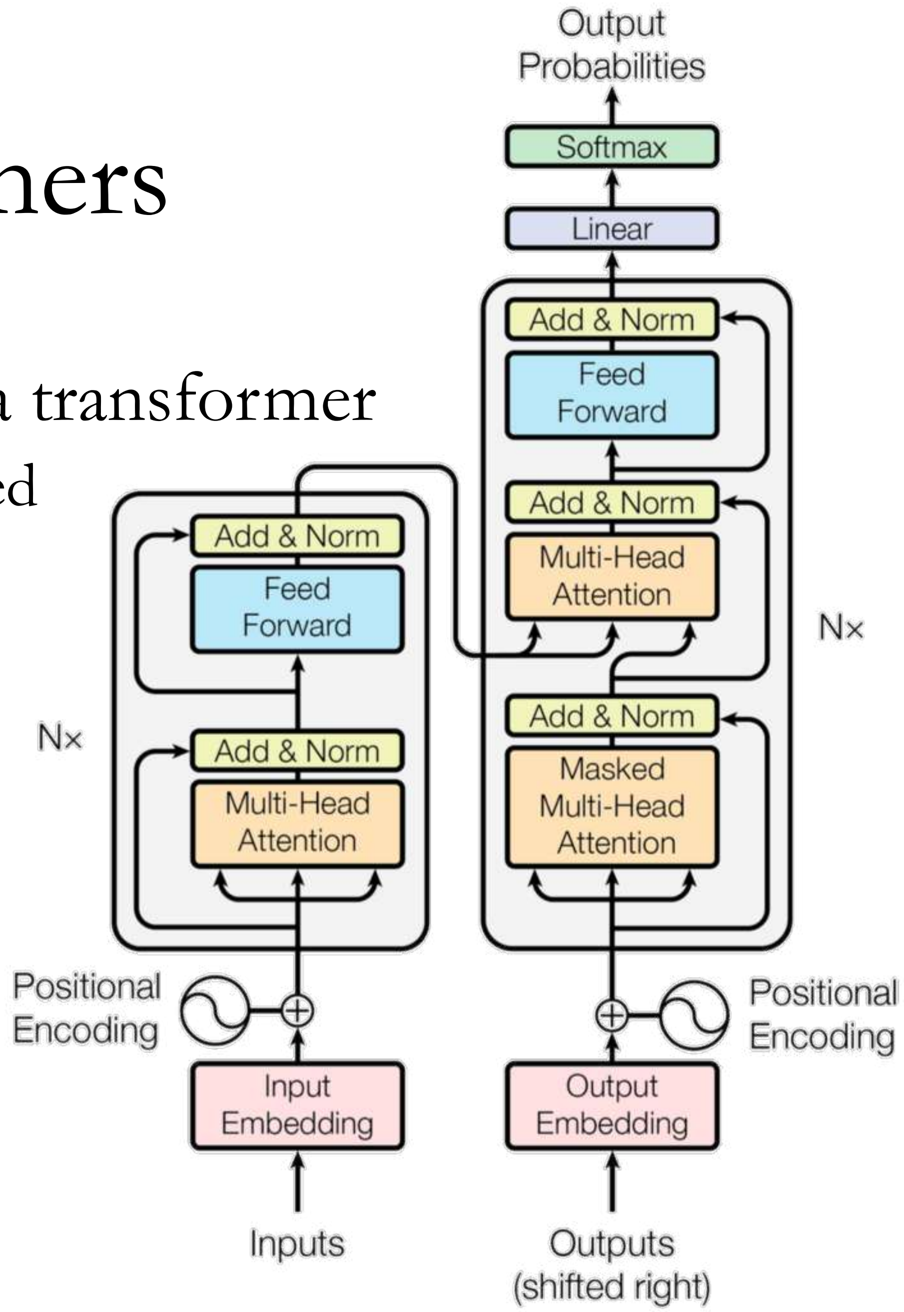


Figure 1: The Transformer - model architecture.

In this lecture, we have covered

- Recurrent language modeling
- Neural machine translation
- Attention mechanism