

# Recurrent neural networks

Materials for this section:

- <https://atcold.github.io/pytorch-Deep-Learning/en/week06/06-3/>
- <https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>
- <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- <https://blog.floydhub.com/attention-mechanism/>

# Recurrent neural networks

Recurrent neural networks (RNNs) are a family of networks specialized for processing sequential data:

$$x(0), \dots, x(T)$$

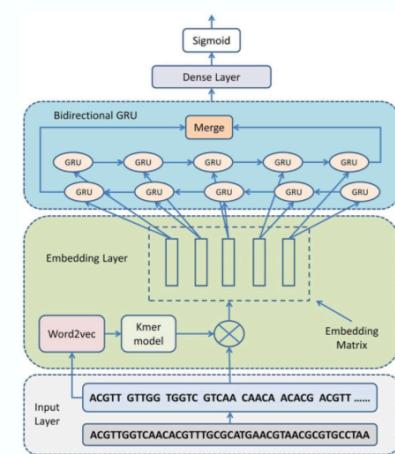
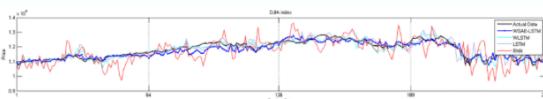
- RNNs can scale to long sequences, and even process sequences of variable length

## Applications:

## Working with Sequential Data

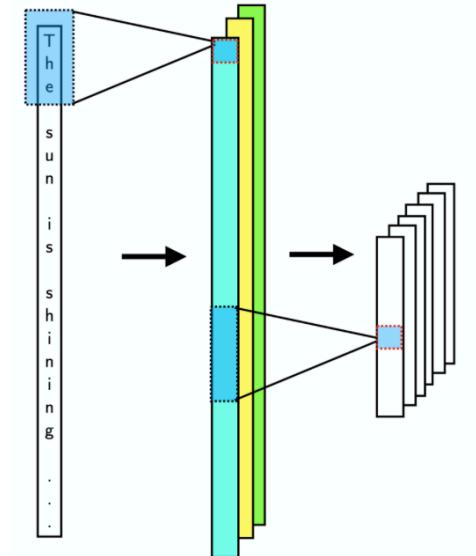
- Text classification
- Speech recognition (acoustic modeling)
- language translation
- ...

## Stock market predictions



DNA or (amino acid/protein)  
sequence modeling

Shen, Zhen, Wenzheng Bao, and De-Shuang Huang. "Recurrent Neural Network for Predicting Transcription Factor Binding Sites." *Scientific reports* 8, no. 1 (2018): 15270.



We can apply CNNs but there is a problem:  
sequences are not i.i.d

- they typically have long-range correlations, for example dependence between words in sentence

## Parameters sharing:

- in convolutional nets we share the same kernel across „**space**”,
- recurrent neural network shares the same weights across several **time** steps, not limited to kernel size as in CNNs.

# Recurrent neural networks

How to capture these correlations?

- by introducing *loops* into networks

Networks we used previously: also called feedforward neural networks

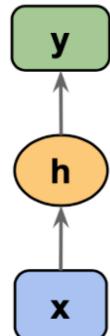
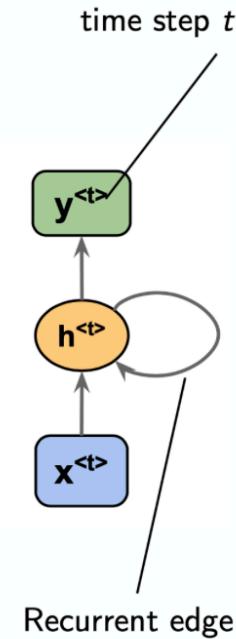


Figure: Sebastian Raschka, Vahid Mirjalili, Python Machine Learning, 3rd Edition, Birmingham, UK: Packt Publishing, 2019

Recurrent Neural Network (RNN)

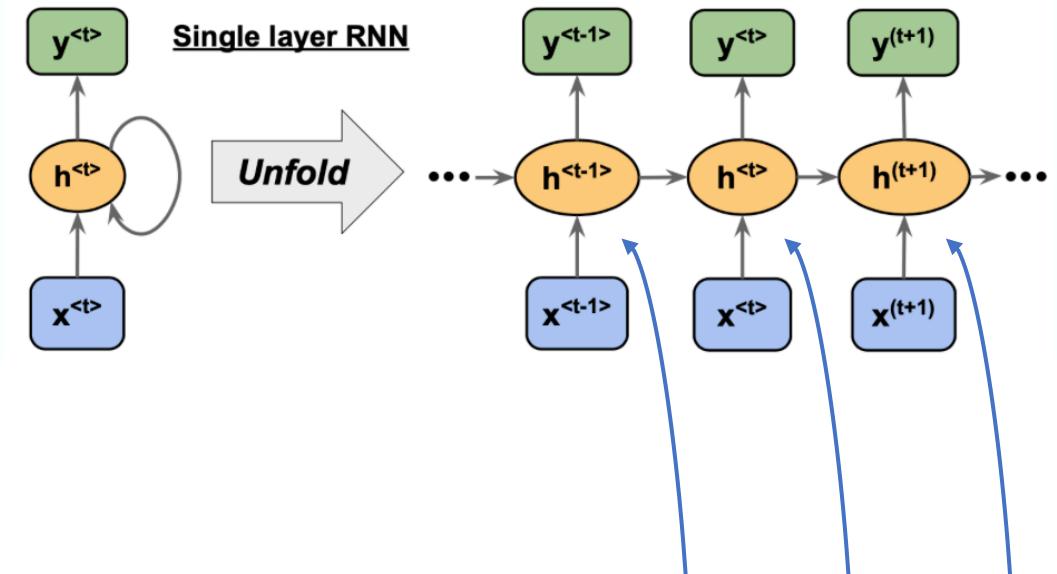


- we process sequence *step-by step*
- hidden state  $h$  depends not only on input at given step, but also on **previous** hidden states

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta),$$

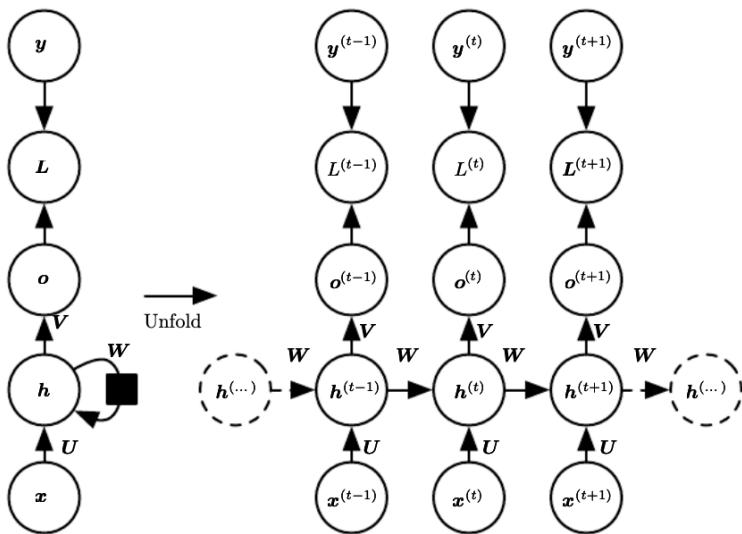
[https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L14\\_intro-rnn/L14\\_intro-rnn-part1\\_slides.pdf](https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L14_intro-rnn/L14_intro-rnn-part1_slides.pdf)

Unroll the loop



parameters (weights)  
are **shared**

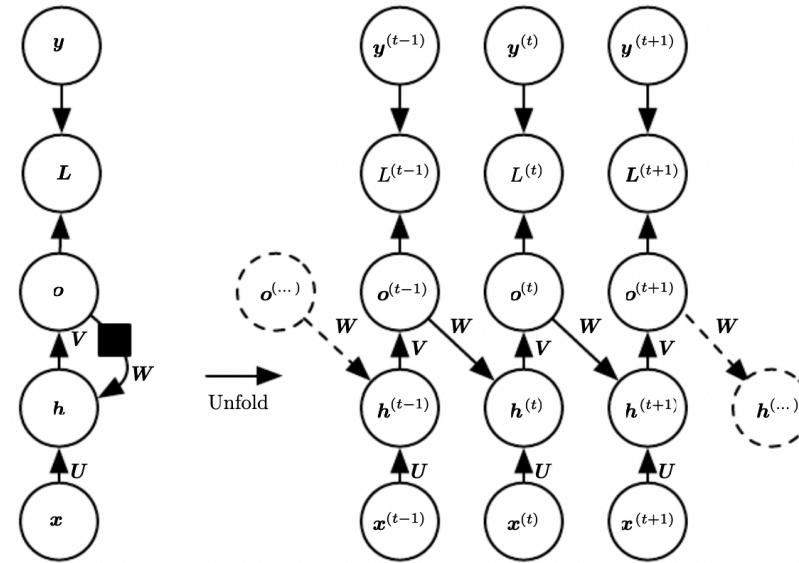
# Recurrent neural networks



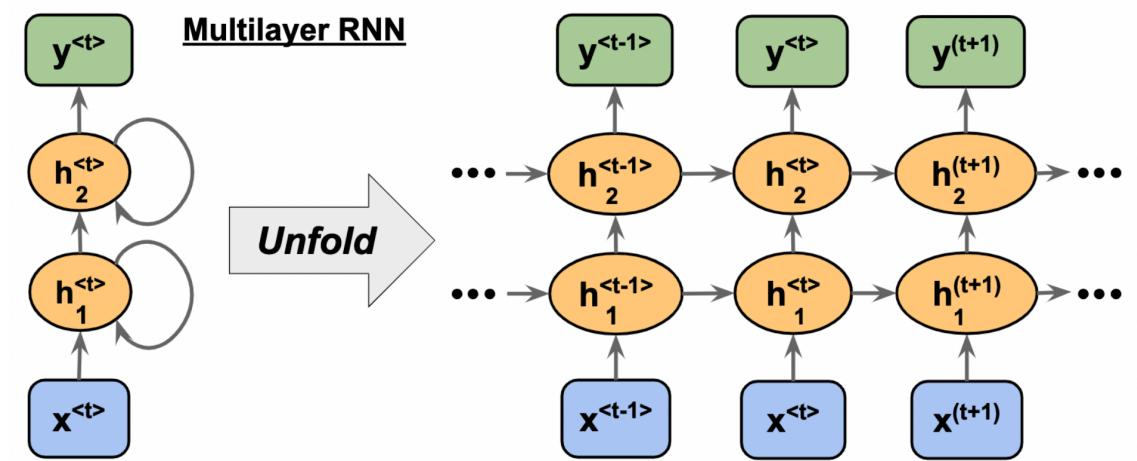
$$\begin{aligned} \mathbf{a}^{(t)} &= \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}, \\ \mathbf{h}^{(t)} &= \tanh(\mathbf{a}^{(t)}), \\ \mathbf{o}^{(t)} &= \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)}, \\ \hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{o}^{(t)}), \end{aligned}$$

Forward propagation starts with  $\mathbf{h}^{(0)}$  initialized *randomly*

- different loop connections

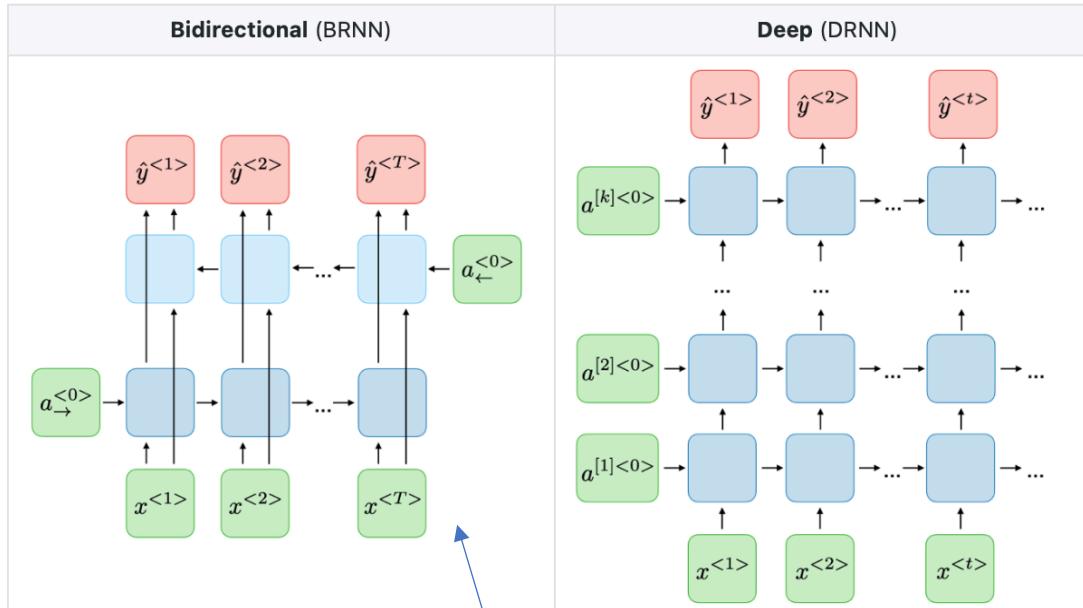


- multiple hidden layers



# Recurrent neural networks

□ **Variants of RNNs** — The table below sums up the other commonly used RNN architectures:



two separate passes

outputs may also affects hidden states  
(*autoregressive network*)

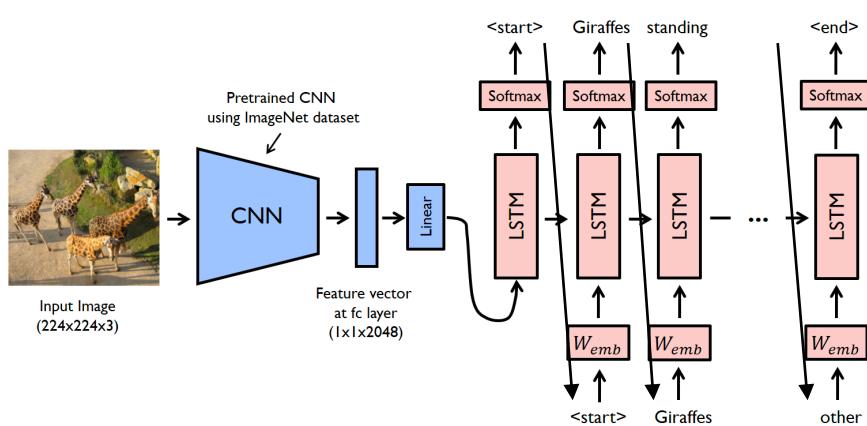
like before

□ **Applications of RNNs** — RNN models are mostly used in the fields of natural language processing and speech recognition. The different applications are summed up in the table below:

Type of RNN	Illustration	Example
One-to-one $T_x = T_y = 1$		Traditional neural network
One-to-many $T_x = 1, T_y > 1$		Music generation
Many-to-one $T_x > 1, T_y = 1$		Sentiment classification
Many-to-many $T_x = T_y$		Name entity recognition
Many-to-many $T_x \neq T_y$		Machine translation

# Recurrent neural networks: applications

Example: image captioning system

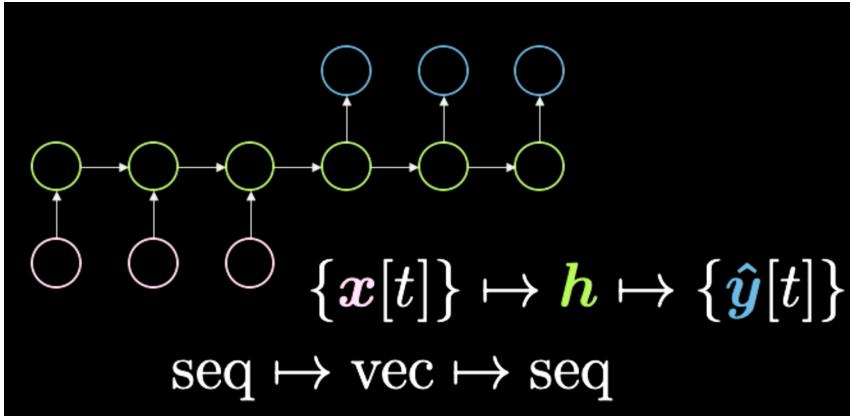


[github.com/anunay999/image\\_captioning\\_vgg16](https://github.com/anunay999/image_captioning_vgg16)



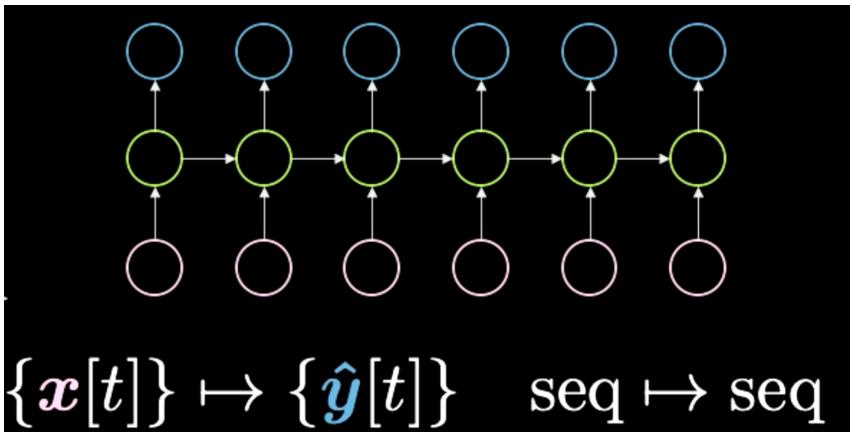
<https://cv-tricks.com/artificial-intelligence/show-attend-tell-image-captioning-explained/>

# Recurrent neural networks: applications



State-of-the-art in machine translation about three years ago (2018), until *Transformers* era

<https://atcold.github.io/pytorch-Deep-Learning/en/week06/06-3/>



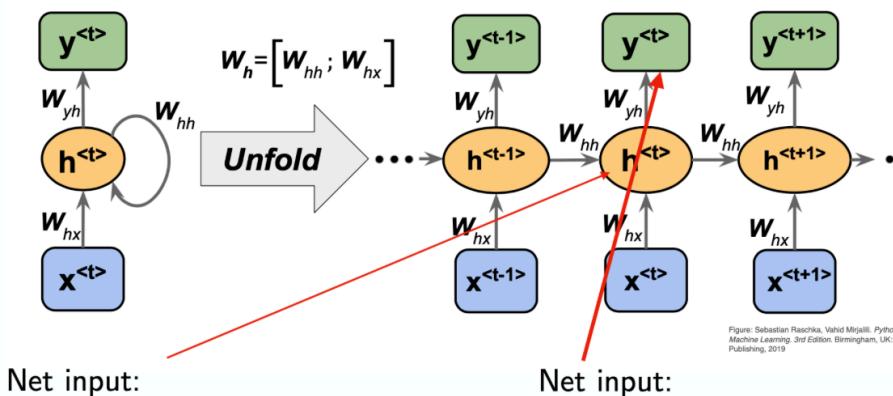
Examples:

- word autocompletion algorithm,
- live streams captioning,
- RNN-writer trained on a corpus of old sci-fi stories:  
<https://www.robinsloan.com/notes/writing-with-the-machine/>

```
test.txt          •      rnn-client.coffee
1 The rings of Saturn glittered while the two men looked at each other.
```

# Recurrent neural networks: training

## Weight matrices in a single-hidden layer RNN



$$\mathbf{z}_h^{(t)} = \mathbf{W}_{hx}\mathbf{x}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h$$

$$\mathbf{h}^{(t)} = \sigma_h(\mathbf{z}_h^{(t)})$$

Sebastian Raschka

STAT 453

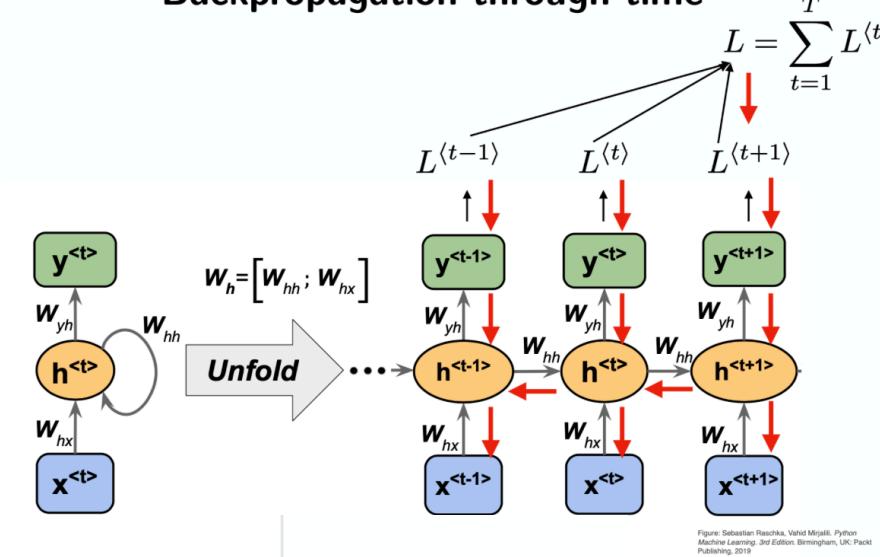
The loss is calculated as sum over all time steps

Training: vanishing/exploding gradients problem  
-> the bigger the longer timeseries is!

$$L = \sum_{t=1}^T L^{(t)} \quad \frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(t)}}{\partial y^{(t)}} \cdot \frac{\partial y^{(t)}}{\partial \mathbf{h}^{(t)}} \cdot \left( \sum_{k=1}^t \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} \cdot \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \right)$$

computed as a multiplication of adjacent time steps:

## Backpropagation through time



$$L = \sum_{t=1}^T L^{(t)}$$

[https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L14\\_intro-rnn/L14\\_intro-rnn-part1\\_slides.pdf](https://github.com/rasbt/stat479-deep-learning-ss19/blob/master/L14_intro-rnn/L14_intro-rnn-part1_slides.pdf)

# Recurrent neural networks: long-range dependencies

Recurrent networks involve the composition of the same function multiple times, once per time step:

$$\mathbf{h}^{(t)} = \mathbf{W}^\top \mathbf{h}^{(t-1)}$$

CHAPTER 10. SEQUENCE MODELING: RECURRENT AND RECURSIVE NETS

and lacking inputs  $\mathbf{x}$ . As described in section 8.2.5, this recurrence relation essentially describes the power method. It may be simplified to

$$\mathbf{h}^{(t)} = (\mathbf{W}^t)^\top \mathbf{h}^{(0)}, \quad (10.37)$$

and if  $\mathbf{W}$  admits an eigendecomposition of the form

$$\mathbf{W} = \mathbf{Q} \Lambda \mathbf{Q}^\top, \quad (10.38)$$

with orthogonal  $\mathbf{Q}$ , the recurrence may be simplified further to

$$\mathbf{h}^{(t)} = \mathbf{Q}^\top \Lambda^t \mathbf{Q} \mathbf{h}^{(0)}. \quad (10.39)$$

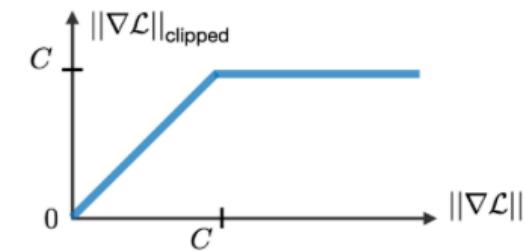
The eigenvalues are raised to the power of  $t$ , causing eigenvalues with magnitude less than one to decay to zero and eigenvalues with magnitude greater than one to explode. Any component of  $\mathbf{h}^{(0)}$  that is not aligned with the largest eigenvector

Problem with learning **long-range** dependencies:

- signal about these dependencies will be hidden by the smallest fluctuations arising from **short-term** dependencies

Exploding gradients are easier to overcome:

□ **Gradient clipping** — It is a technique used to cope with the exploding gradient problem sometimes encountered when performing backpropagation. By capping the maximum value for the gradient, this phenomenon is controlled in practice.



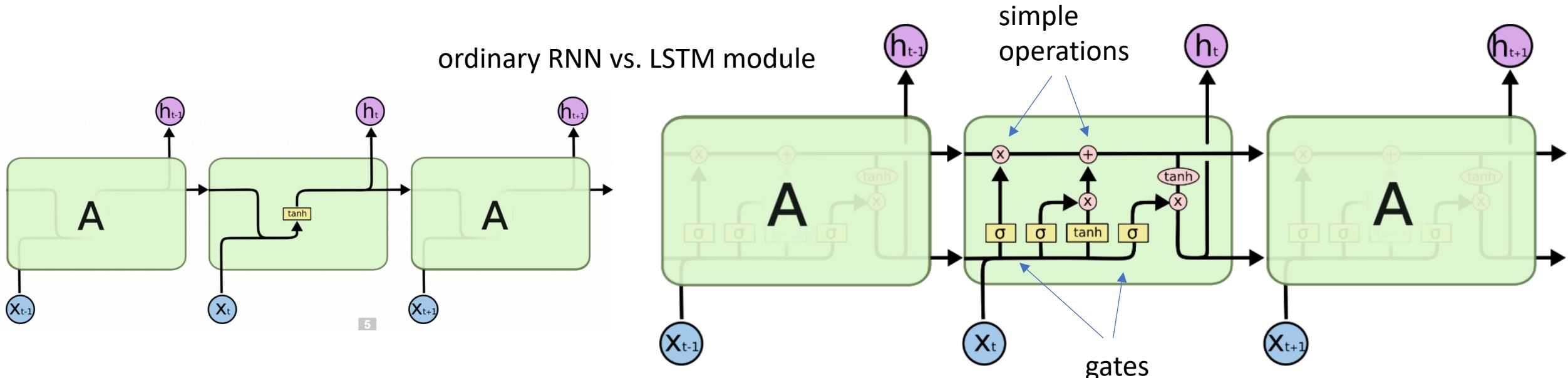
Solution: **gated RNNs**:

- **Long short-term memory (LSTM)** -- uses a memory cell for modeling long-range dependencies and avoid vanishing gradient problems

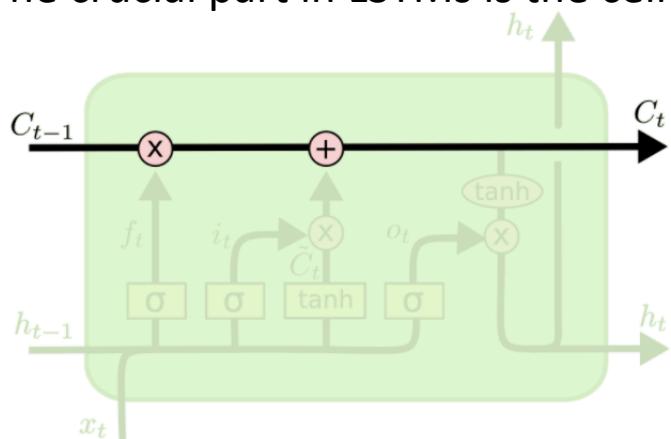
Hochreiter, Sepp, and Jürgen Schmidhuber. "[Long short-term memory](#)." *Neural computation* 9, no. 8 (1997): 1735-1780.

- Gated recurrent unit (GRU)

# Recurrent neural networks: LSTM nets

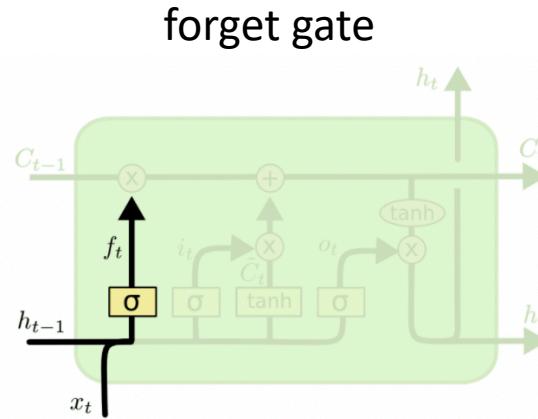


The crucial part in LSTMs is the cell state

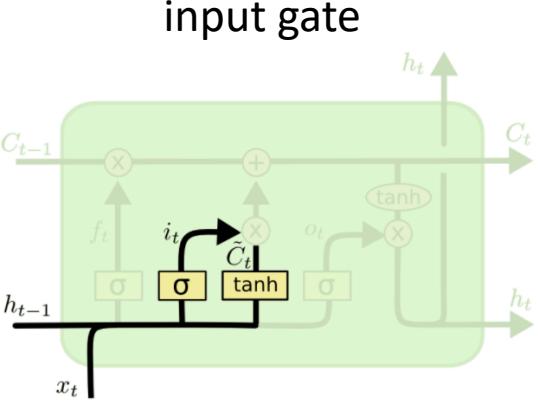


- gates regulate what amount of information is preserved or removed from the cell state

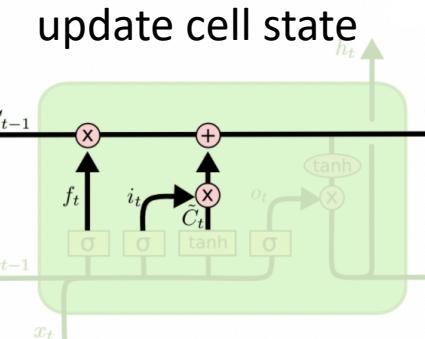
# Recurrent neural networks: LSTM nets



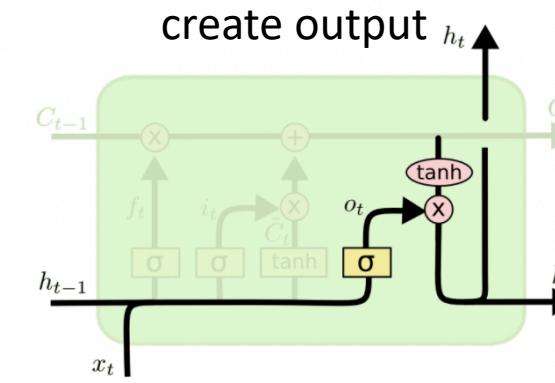
$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$



$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$



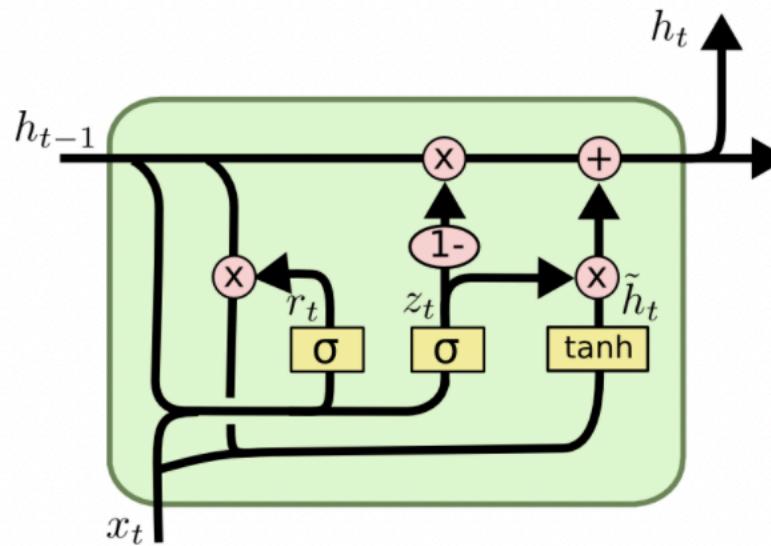
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



$$o_t = \sigma (W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

# Recurrent neural networks: LSTM nets

- LSTM simplification: GRU



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

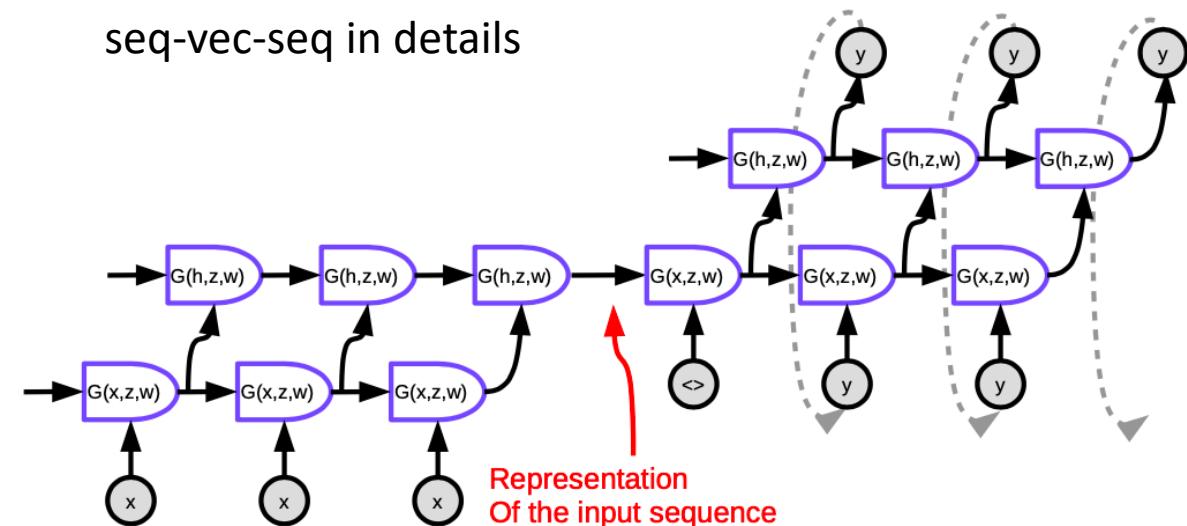
$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- GRU sometimes performs better than LSTM sometimes not
- Although LSTMs/GRUs are still widely used (e.g. in NLP), their popularity is decreasing

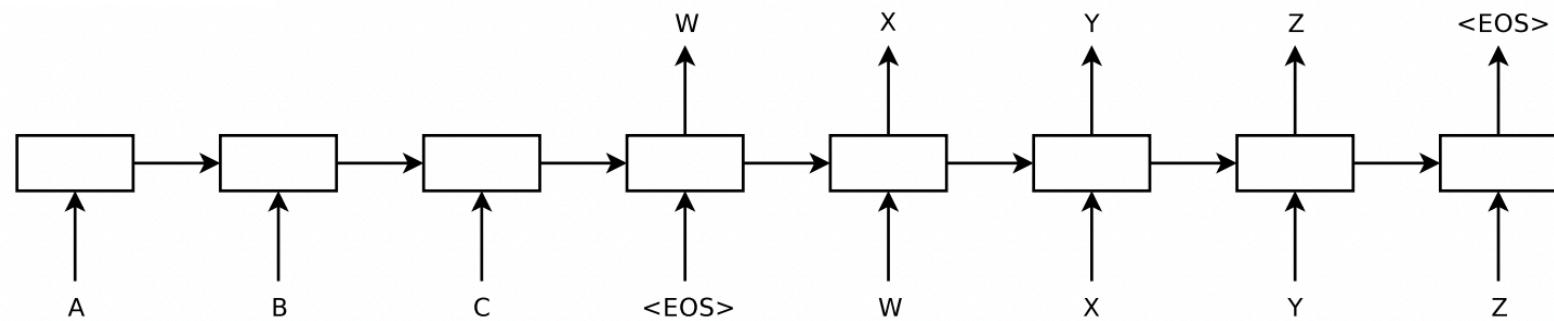
# Attention in neural nets

seq-vec-seq in details



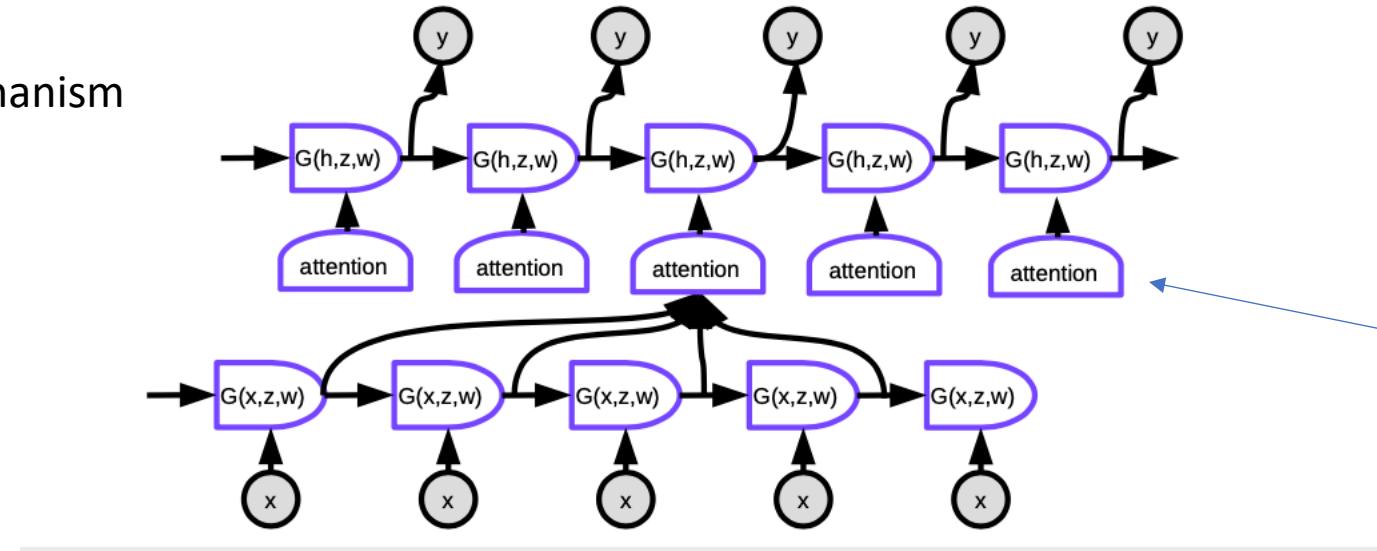
Unfortunately, even using LSTMs,  
such model does not preserve  
information for more than 20 words

The approach proposed by Sutskever in 2014 is the first neural machine translation system to have comparable performance to classic approaches. It uses an *encoder-decoder* architecture where both the encoder and decoder are multi-layered LSTMs.



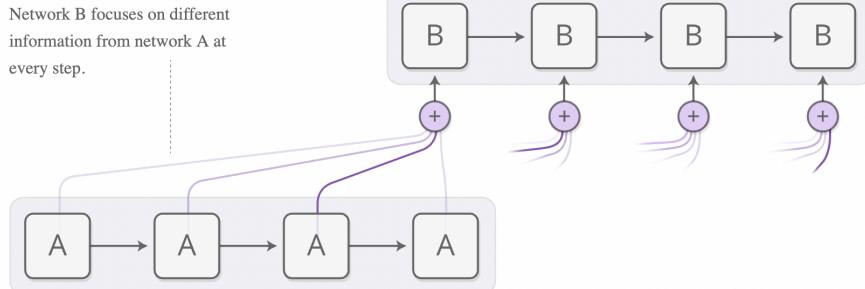
# Attention in neural nets

## Attention mechanism

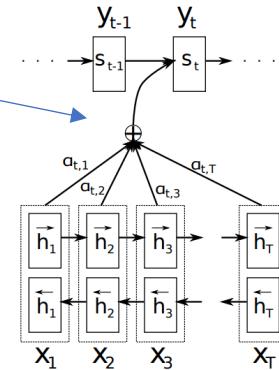


Neural networks can achieve better behavior using *attention*, focusing on **wider subset** of the information at each step

- RNN attend over the output of another RNN: at every time step, it focuses on different positions.



separate attention scores at each time step



The context vector  $c_i$  is, then, computed as a weighted sum of these annotations  $h_i$ :

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \quad (5)$$

The weight  $\alpha_{ij}$  of each annotation  $h_j$  is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}, \quad (6)$$

where

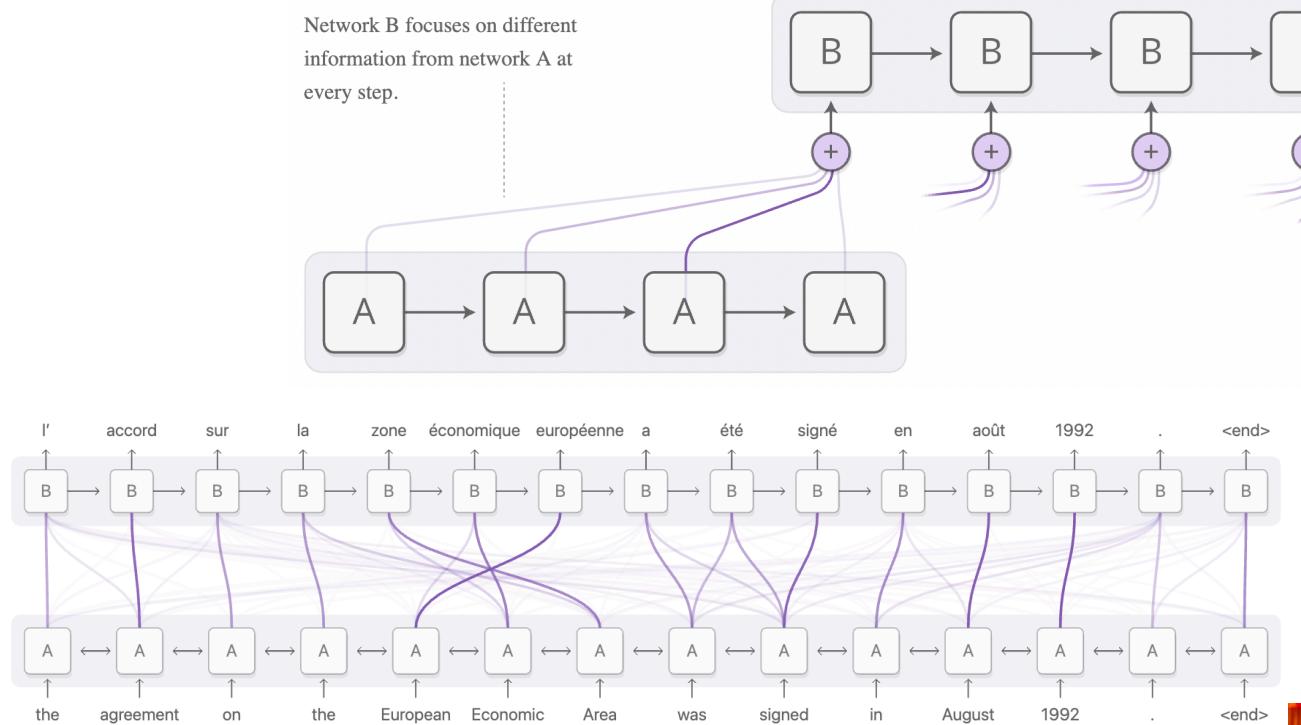
$$e_{ij} = a(s_{i-1}, h_j)$$

is an *alignment model* which scores how well the inputs around position  $j$  and the output at position  $i$  match. The score is based on the RNN hidden state  $s_{i-1}$  (just before emitting  $y_i$ , Eq. (4)) and the  $j$ -th annotation  $h_j$  of the input sentence.

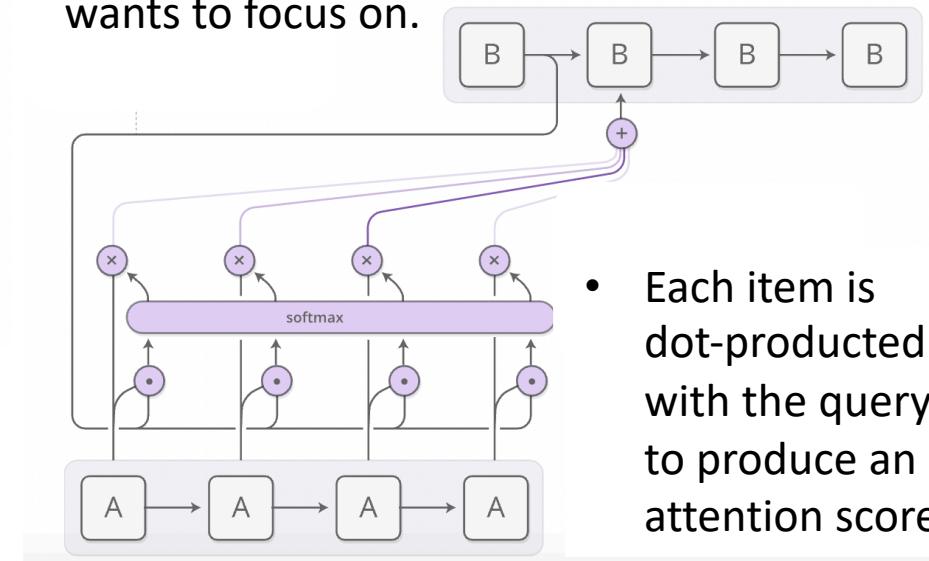
# Attention in neural nets

How to generate attention distribution?

-> *content-based* attention



- The attending RNN generates a query describing what it wants to focus on.

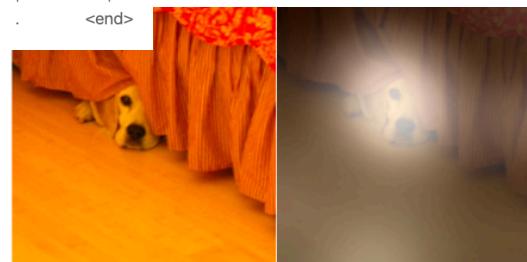


- Each item is dot-produced with the query to produce an attention score

- When generating the output we focus on words as they become relevant



A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



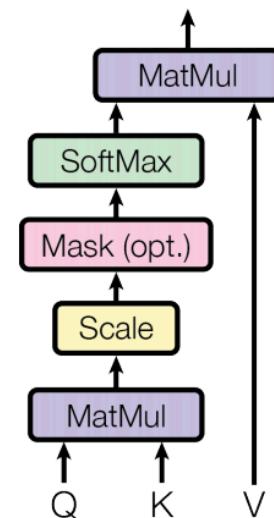
A stop sign is on a road with a mountain in the background.

# Attention in neural nets

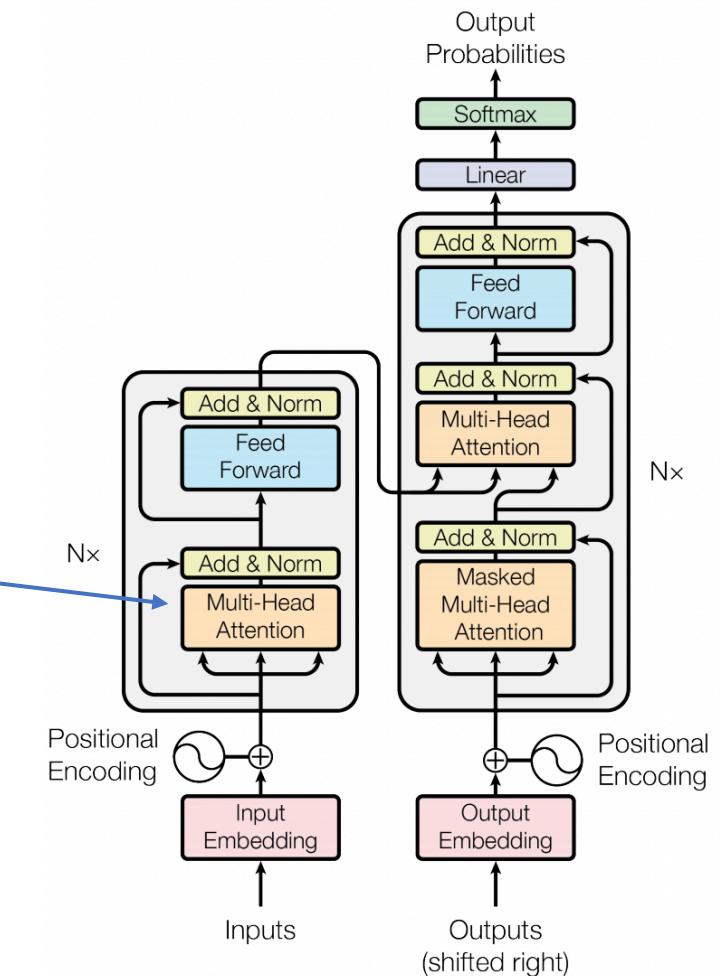
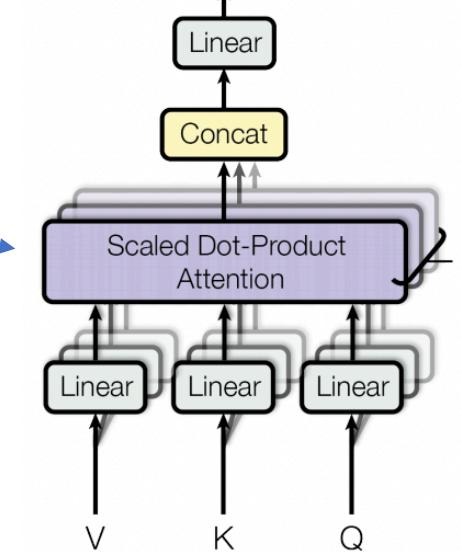
## Transformer architecture

- utilize *self-attention*
- just feed-forward connections, without recurrence

Scaled Dot-Product Attention



Multi-Head Attention



Notable transformers:

- **BERT** (2018): SOTA in machine translations
- **DETR** (2020): object detection with transformers

Figure 1: The Transformer - model architecture.