

# Recurrent Neural Networks and Advanced Architectures for NLP

Jelke Bloem & Giovanni Colavizza

Text Mining  
Amsterdam University College

March 28, 2023

# Announcements

- Assignment 3 deadline: Friday 14th of April
- Talk about projects on Friday
- BERT reading assignment: Tue 18th of April
- Project update 0: Thu 20th of April

# Overview

- 1 Neural Language Models
- 2 Recurrent Neural Networks
- 3 Fancy RNNs

## Questions/discussion Transformers reading assignment

- How would a Transformer deal with translating a whole book? Are there any available models that can deal with this type of task as good as (or even better than) humans do?
- The paper mentions that the hyperparameters of the Transformer model were extensively fine-tuned. But in class we have learned that, once we apply a model on our test data, we cannot test it on the same data again. How did the authors manage to find enough data to train it so extensively, and did they reuse some of it?
- Often, all-purpose decision are good at everything and excellent at nothing. Seemingly, it was not the case with transformers. However, I still suspect there should be tasks that the transformers are not a preferred model choice, are there any?
  - ▶ Redundant attention heads, downsize for inference
  - ▶ No recursive computation
- What other programs use self-attention? Is it something that could be implemented in more things to improve them?
- If the Transformer model has been adapted to work for non-textual data, does it outperform other models for these forms of data?

# Questions/discussion Transformers reading assignment: Explainable AI

- How would transformer models like this deal with languages that have completely different lexical structures? For example, translating English to Arabic?
- How reliable are these in deployment when even the makers of such models cannot explain what actually happens at every stage of the process.
- Based on the sentence “individual attention heads clearly learn to perform different tasks; many appear to exhibit behavior related to the syntactic and semantic structure of the sentences.” — what and how this “behavior” is exactly evaluated? The authors mention this in context of “interpretable models”, but does this interpretation takes act on the user-end or during the training stage?
- Why do they say self-attention could yield more interpretable models?

# Questions/discussion Transformers reading assignment

- How did they make a whole model out of a single mechanism which is otherwise only used for upgrading an existing model?
  - ▶ No more recurrence or convolutions, but still feedforward NN.
- Why weren't transformers explored earlier?
- After transformers, improvement because of innovations in the field at a theoretical level, or just due to increased model complexity and training data?

# Neural Language Models

# Recap

With language models, we want to compute:

- The probability of a sequence of words:  $P(w) = P(w_1, w_2, \dots, w_n)$ .
- The probability of a new word given what came before it:  
 $P(w_n | w_1, w_2, \dots, w_{n-1})$ .
- We have also seen n-gram language models which make use of the Markov assumption. For example, a trigram language model would predict the probability of a sequence of words by conditioning on the two previous words at each step:

$$P(w) = P(w_1)P(w_2|w_1)P(w_3|w_1, w_2) \dots P(w_n|w_{n-2}, w_{n-1})$$

- n-gram language models have some issues, including sparsity and limited use of context.

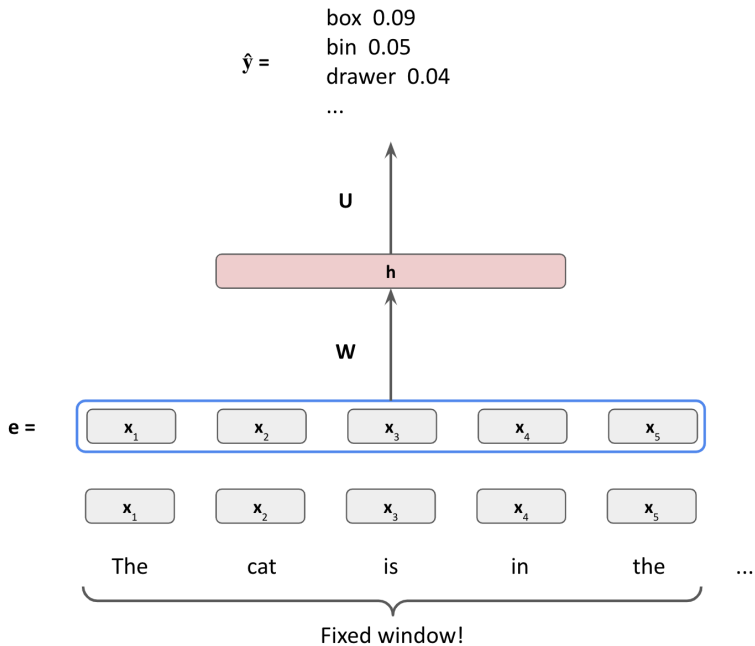


# A first neural language model

We can start by using word embeddings as features for a **fixed-window neural language model**:

- Given a fixed-window sequence of words represented using their embeddings  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t$ ;
- we are interested in estimating the probability of the next word  $\hat{p}(\mathbf{x}_{t+1} | \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_t)$ .
- We can use the previous words' embeddings as features, concatenate them and feed them to a hidden layer with a non-linearity, and concluding by estimating probabilities with a softmax.

# A first neural language model



# A first neural language model

Where:

- Each  $\mathbf{x}$  is a  $1 \times d$  embedding vector of dimensionality  $d$ .
- $\mathbf{e} = [\mathbf{x}_1; \mathbf{x}_2; \dots; \mathbf{x}_t]$  is the concatenation of the embeddings of the words in the fixed-window preceding  $w_{t+1}$ . Therefore  $\mathbf{e}$  has dimensionality  $1 \times dk$ , where  $k$  is the number of words in the fixed-window.
- $\mathbf{h} = f(\mathbf{W}\mathbf{e}^T)$  is the hidden layer, with  $f$  an appropriate activation function and  $\mathbf{W}$  a weight matrix.  $\mathbf{W}$  has dimensionality  $dk \times h$  and  $\mathbf{h}$  has dimensionality  $1 \times h$ . The dimensionality of the embeddings and the hidden layer are hyperparameters of the model.
- Finally,  $\hat{y} = \text{softmax}(\mathbf{U}\mathbf{h}^T)$  is the predicted next word probability as estimated using a softmax function. Here  $\mathbf{U}$  is another weight matrix of dimensionality  $|V| \times h$ , so that with the softmax we predict a probability distribution over the vocabulary  $V$ .

# A first neural language model

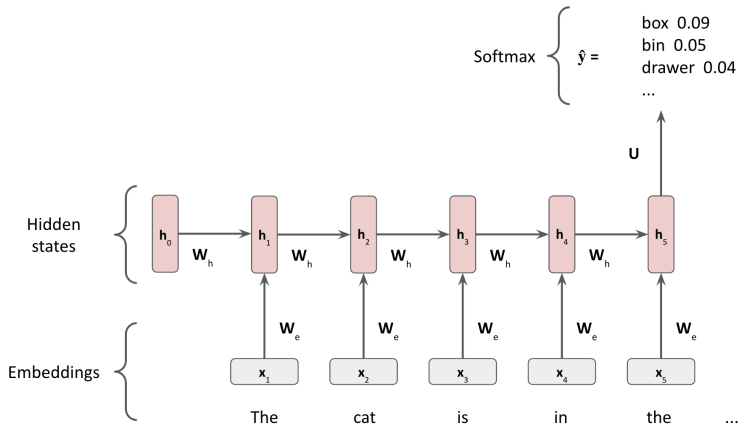
This model is interesting, yet unfortunately:

- it does not solve the use of a broader context with flexible word windows;
- larger windows would mean more parameters so scale is also an issue;
- and it does not make an efficient use of parameters and shared weights.

Can we do better? Yes, with **Recurrent Neural Networks**.

# Recurrent Neural Networks

# Recurrent architecture



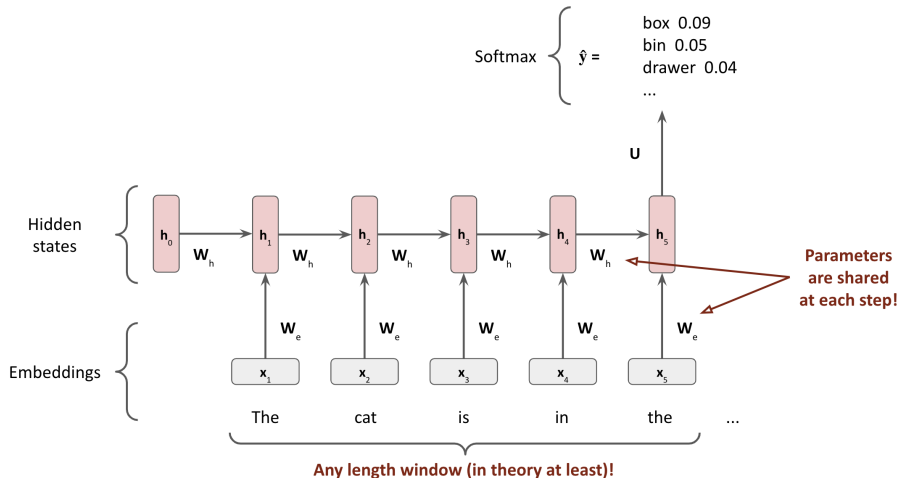
# Recurrent architecture

Where:

- Each  $\mathbf{x}$  is a  $1 \times d$  embedding vector of dimensionality  $d$ .
- $\mathbf{h}_t = f(\mathbf{W}_h \mathbf{h}_{t-1}^T + \mathbf{W}_e \mathbf{x}_t^T)$  is the hidden layer, with  $f$  an appropriate activation function,  $\mathbf{W}_e$  and  $\mathbf{W}_h$  weight matrices.  $\mathbf{W}_h$  has dimensionality  $h \times h$ ,  $\mathbf{W}_e$  has dimensionality  $h \times d$ , and  $\mathbf{h}$  has dimensionality  $1 \times h$ .  $\mathbf{h}_0$  is the initial hidden layer. The dimensionality of the embeddings and the hidden layer are hyperparameters of the model.
- Finally,  $\hat{y} = \text{softmax}(\mathbf{U} \mathbf{h}^T)$  is the predicted next word probability as estimated using a softmax function. Here  $\mathbf{U}$  is another weight matrix of dimensionality  $|V| \times h$ , so that with the softmax we predict a probability distribution over the vocabulary  $V$ .

(Notebook 8.1 Part II: Model)

# Recurrent architecture





# Why RNNs?

RNNs are a big step forward re. our previous concerns:

- Can process inputs on any length and use previous context of any length (in theory);
- model size does not depend on window size ( $W$  matrices remain of the same dimension);
- weights are shared across time steps.

Nevertheless, RNNs can be slow and won't really remember information from many steps back.

# Training RNNs

How can we train RNNs?

- Predict the next word at each step, and calculate the loss accordingly.
- The loss at step  $t$  is the usual **cross-entropy** (now on multiple classes), calculated for the word to be predicted at  $t + 1$ :

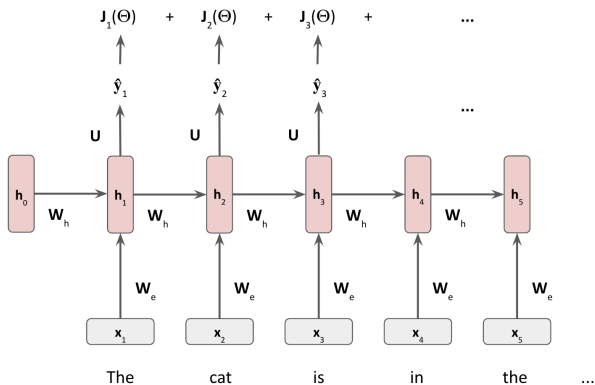
$$\mathcal{L}_t(\mathbf{W}) = - \sum_{w \in V} y_{w_{t+1}} \log(\hat{y}_{w_{t+1}}) = -\log(\hat{y}_{w_{t+1}})$$

- The overall loss is the average of the sum of the losses, calculated at each step:

$$\mathcal{L}(\mathbf{W}) = \frac{1}{T} \sum_{t=1}^T \mathcal{L}_t(\mathbf{W}) = \frac{1}{T} \sum_{t=1}^T -\log(\hat{y}_{w_{t+1}})$$

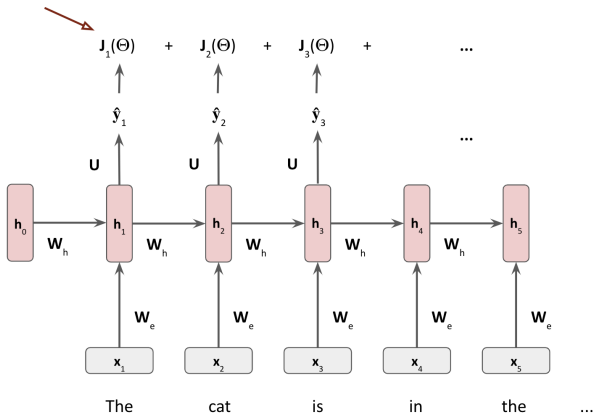
- Optimization can be done via SGD (backpropagation). Since the parameters  $\mathbf{W}$  are used repeatedly, this is sometimes called **backpropagation through time**.

# Training RNNs



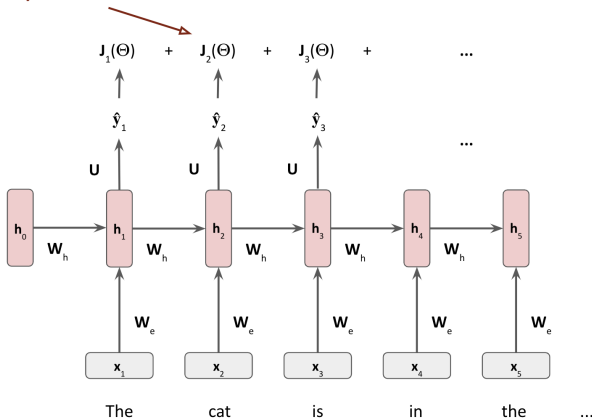
# Training RNNs

Negative log  
probability for "cat"



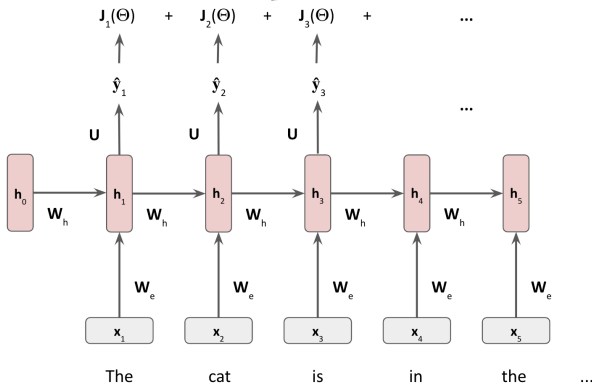
# Training RNNs

Negative log  
probability for "is"



# Training RNNs

Negative log  
probability for "in"



## Recall perplexity?

A second look at perplexity:

- We defined it as the inverse probability of the corpus, normalized by the number of words. For a corpus composed of  $n$  words:

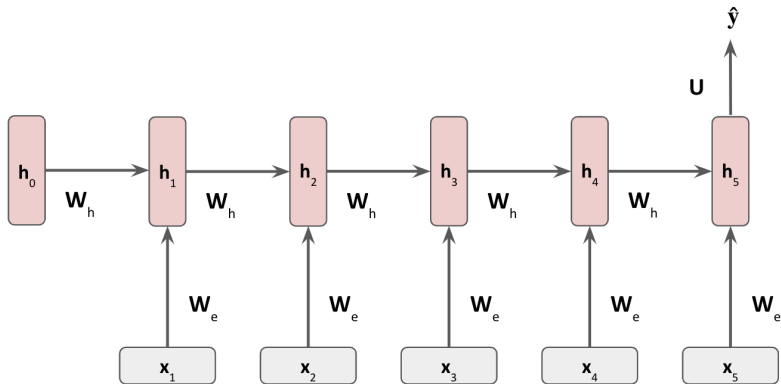
$$PP = P(w_1, w_2, \dots, w_n)^{-\frac{1}{n}} = \prod_{i=1}^n \left( \frac{1}{P(w_i | w_1, \dots, w_{i-1})} \right)^{\frac{1}{n}}$$

- It is actually equal to the exponential of the cross-entropy loss:

$$PP = \prod_{i=1}^n \left( \frac{1}{\hat{y}_{w_{i+1}}} \right)^{\frac{1}{n}} = \exp \left( \frac{1}{n} \sum_{i=1}^n -\log(\hat{y}_{w_{i+1}}) \right) = \exp(\mathcal{L})$$

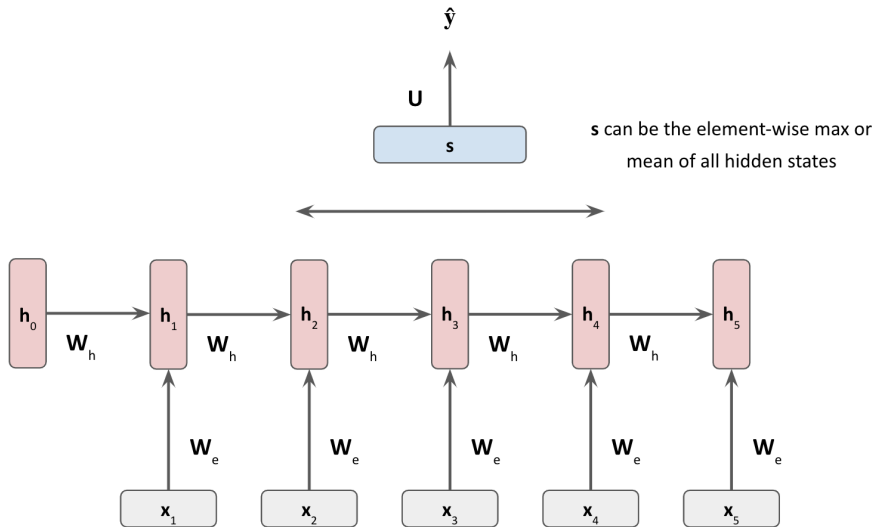
- So *lower perplexity == lower loss == higher data likelihood*.

## RNN flavors: Many to one

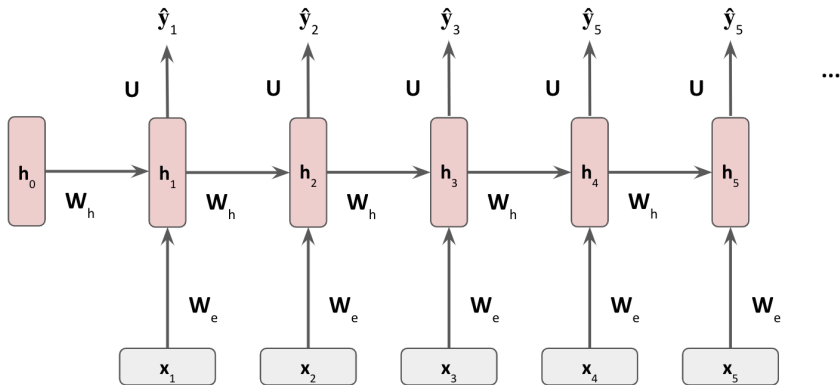




## RNN flavors: Many to one



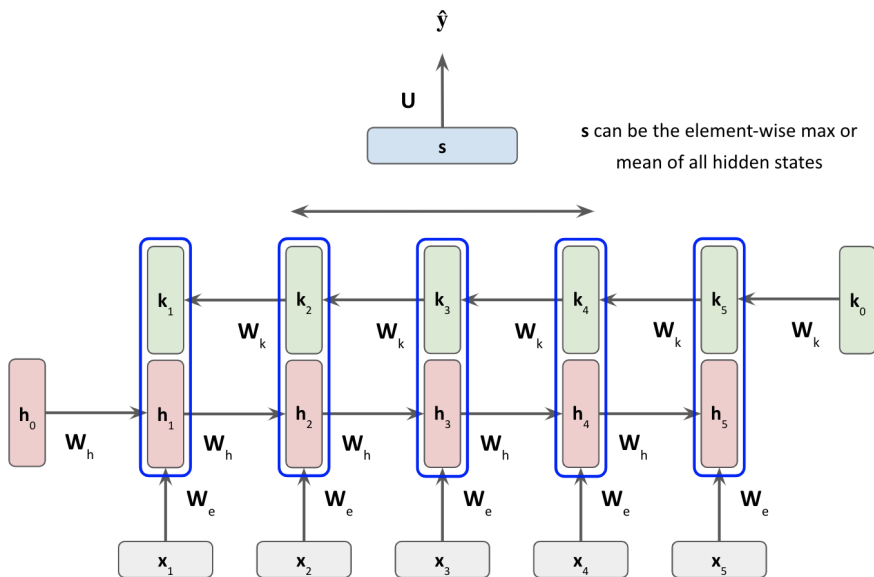
## RNN flavors: Many to many



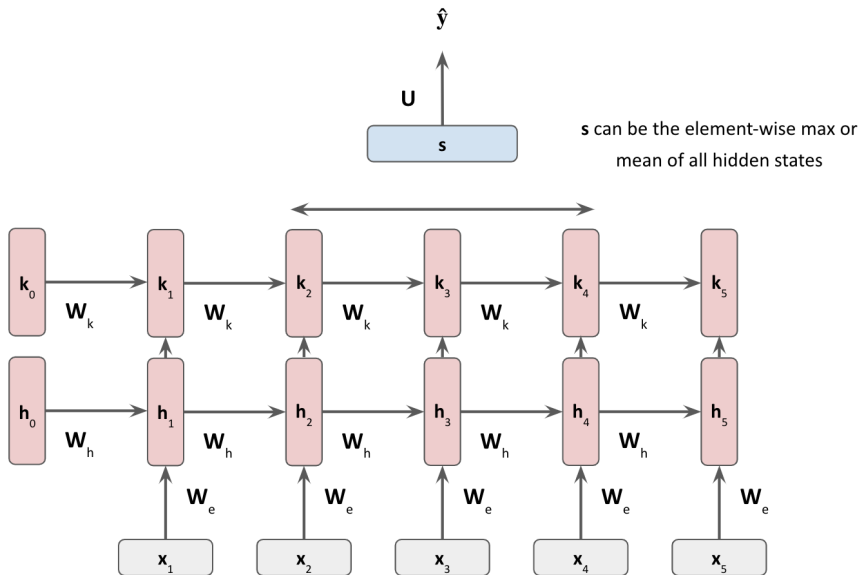
# Notes

## Fancy RNNs

# Bi-RNNs

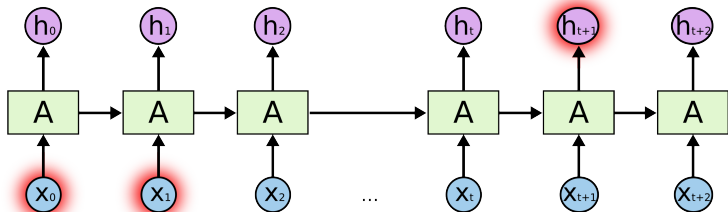


# Multilayer RNNs



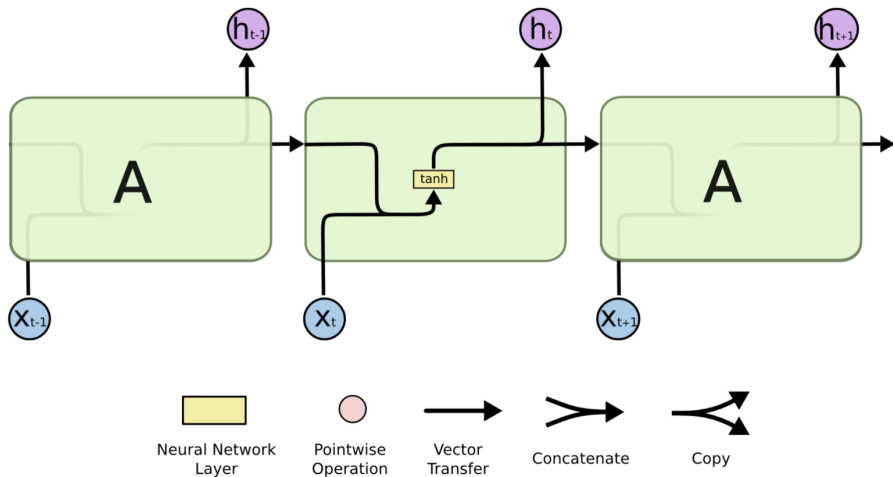
# Vanishing Gradients

- RNNs have a crucial issue: **vanishing and exploding gradients**.
- Both occur when we backpropagate through time with multiplying several times by  $W$ . If  $W$ 's parameters are small, gradients can vanish to zero. If they are large, they can explode.
- This is an issue as it does not allow to model far away context (vanishing) or to properly converge (exploding).
- Solutions:
  - 1 Exploding: **gradient clipping**.
  - 2 Vanishing: **Long Short-Term Memory** networks.



Credit: <https://colah.github.io/posts/2015-08-Understanding-LSTMs>

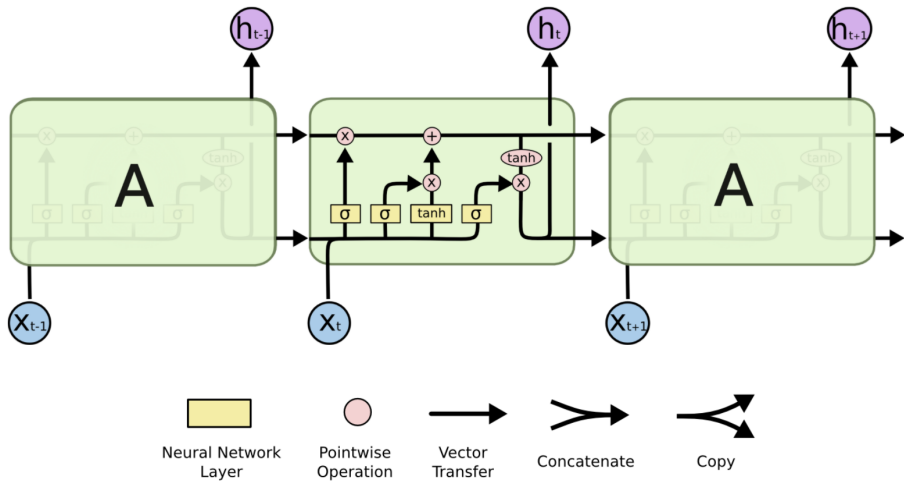
## A different view on RNNs



Credit: <https://colah.github.io/posts/2015-08-Understanding-LSTMs>

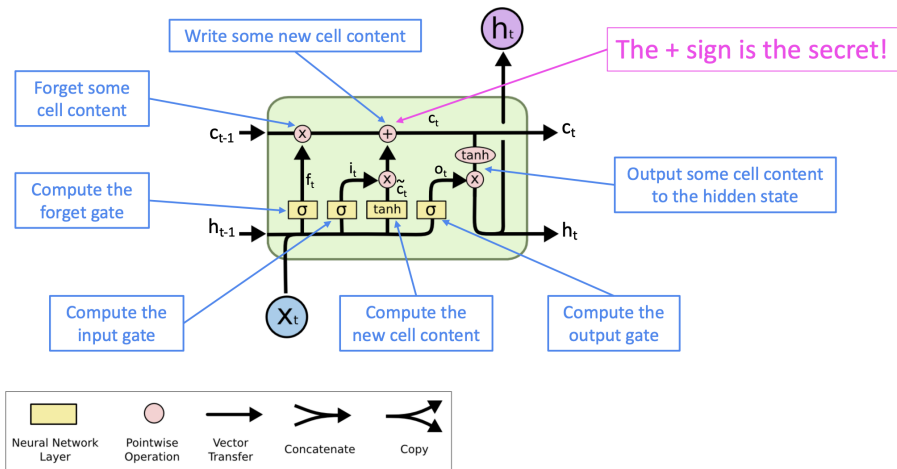


# LSTMs



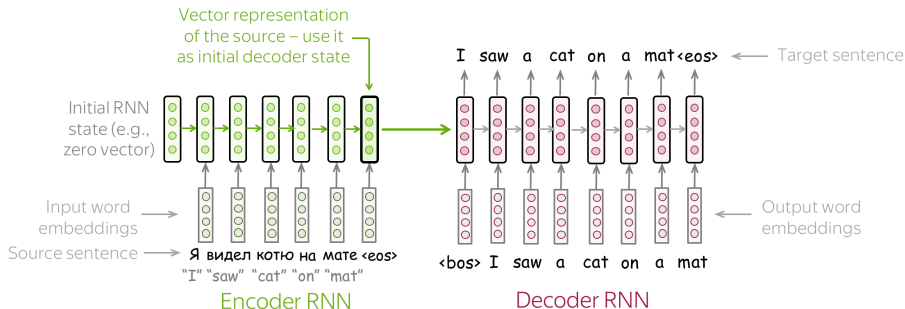
Credit: <https://colah.github.io/posts/2015-08-Understanding-LSTMs>

# LSTMs



*Credit: Stanford CS224N*

# Seq2Seq



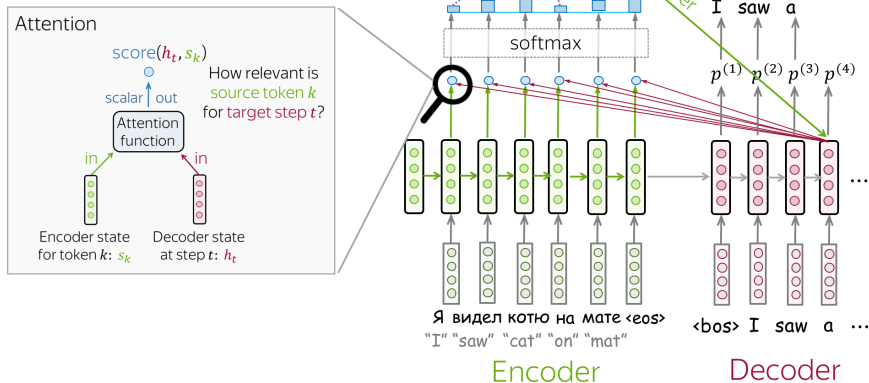
Credit: Lena Voita [https://lena-voita.github.io/nlp\\_course.html](https://lena-voita.github.io/nlp_course.html)

# Attention

Attention output: weighted sum of encoder states with attention weights

Attention weights: distribution over source tokens

A model can learn to “pay attention” to the most relevant source tokens for each step



Credit: Lena Voita [https://lena-voita.github.io/nlp\\_course.html](https://lena-voita.github.io/nlp_course.html)

## What's next

- The most recent advances in neural networks for NLP come from the shift from recurrent architectures to using **attention-based architectures**.
- For your reading assignments, you have explored **transformers** (which combine attention with other ideas from neural networks literature) and will explore **BERT** (which is a neural language model making use of transformers).
- While there is much more to it, in this way you will have a window into contemporary models for NLP.

The next part of the course turns to other topics instead: Web scraping and APIs, recommender systems, corpus annotation, sentiment analysis and clustering with topic modelling, ethics.

# Notes