

New task: Machine Translation

Machine translation tackles the problem of translating a sentence x from the source language to the target language y .

Source language:

On ciągle myśli o niebieskich migdałach.

Target language:

He keeps thinking about blue almonds?

He constantly thinks about blue almonds?

NMT as a flagship task of NLP

NMT continues to thrive

- NMT researched has pioneered many of the recent innovations of NLP
 - attention
 - subwords
 - self-attention
 - unsupervised learning
 - and many more

Unsupervised NMT

- The parallel corpora are still scarce
- However, in any language there is an abundance of monolingual data that is freely available
- Research on unsupervised MT focuses on eliminating the dependency on labeled data, which is especially beneficial for low-resource languages

Cross-lingual Word Embeddings

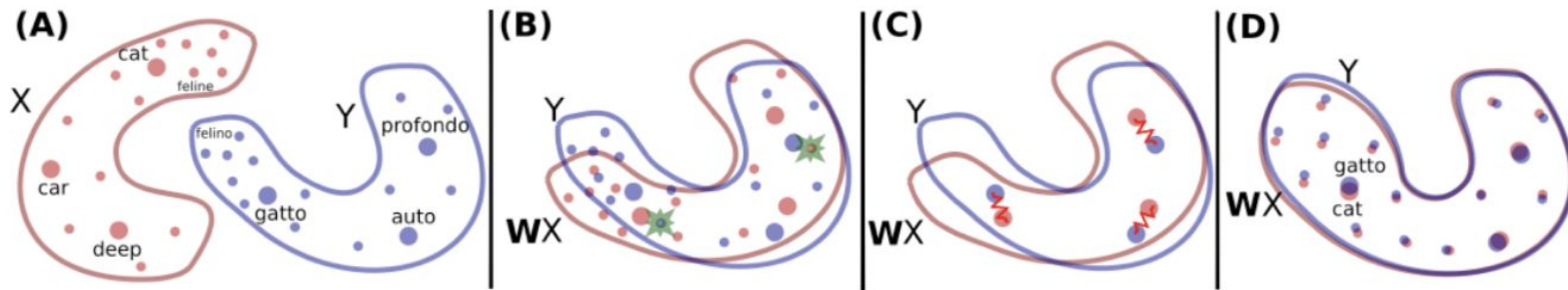
- Idea:

Cross-lingual Word Embeddings

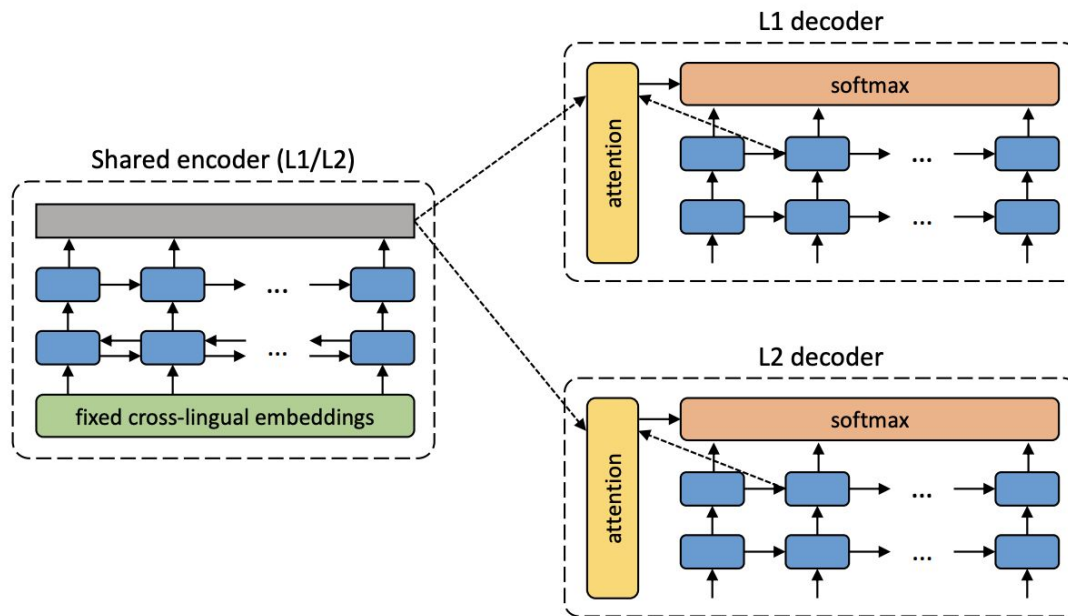
- Idea: Vector spaces representing words in different languages can be aligned, because words with similar meanings have similar statistical properties regardless of the language
- Many current CLE methods map two monolingual distributions of words to a shared vector space using linear transformations

Cross-lingual Word Embeddings

- Idea: Vector spaces representing words in different languages can be aligned, because words with similar meanings have similar statistical properties regardless of the language
- Many current CLE methods map two monolingual distributions of words to a shared vector space using linear transformations

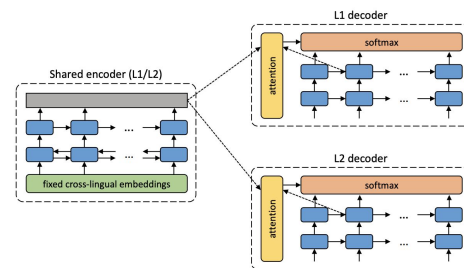


Unsupervised NMT [Artetxe et al., 2017]



Shared encoder for two languages

- Proposed system makes use of one and only one encoder that is shared by both languages involved
- This universal encoder is aimed to produce a language independent representation of the input text, which each decoder should then transform into its corresponding language



Denoising step

- Encode the sentence using the shared encoder
- Reconstruct the original sentence using the decoder of that language
- Problem:

Denoising step

- Encode the sentence using the shared encoder
- Reconstruct the original sentence using the decoder of that language
- Problem: Lots of degenerated solutions by trivial copying
- The system would at best make very literal word-by-word substitutions when used to translate from one language to another at inference time
- Solution: Introduce random noise

On-the-fly backtranslation

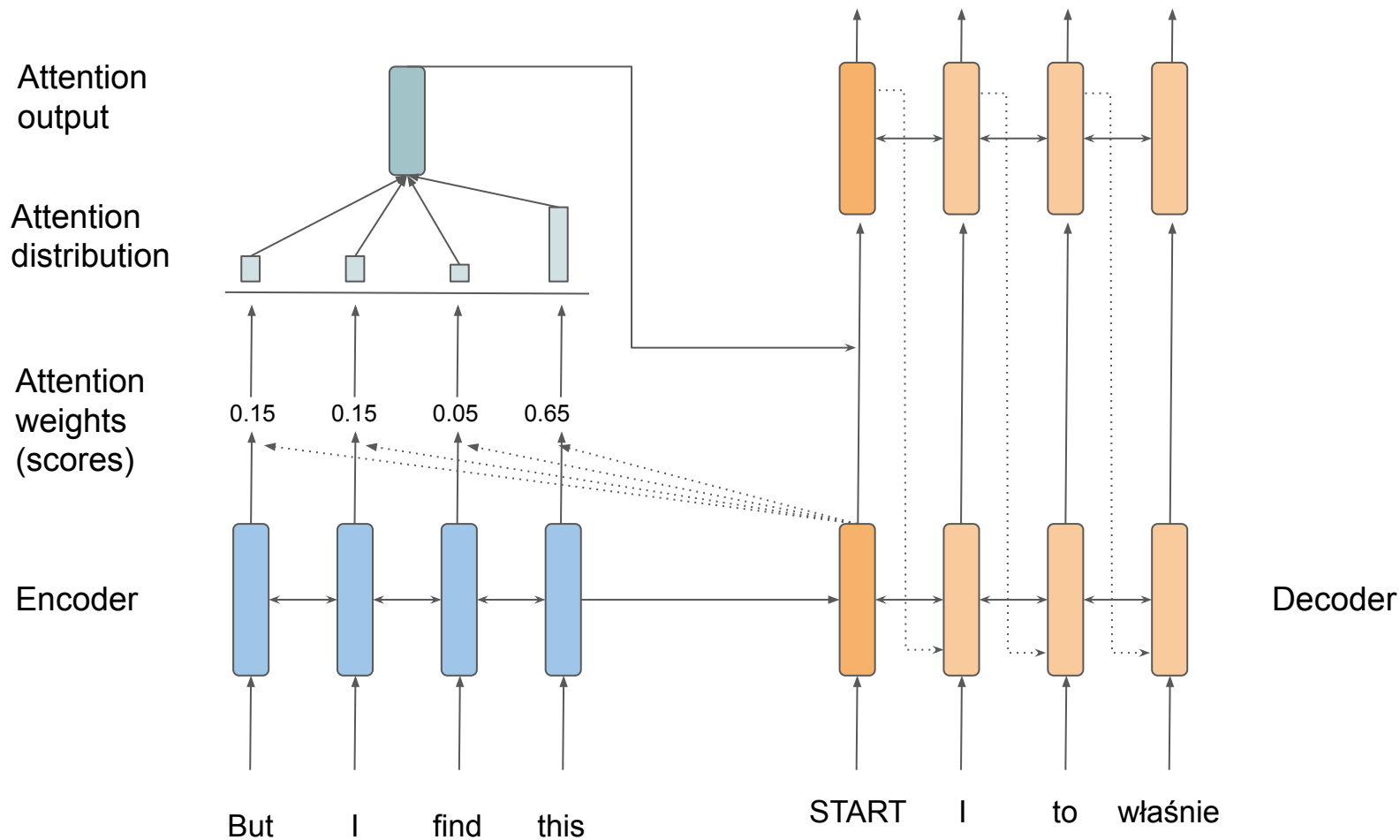
- Given an input sentence in one language, we use the system in inference mode with greedy decoding to translate it to the other language
- We obtain a pseudo-parallel sentence pair, and train the system to predict the original sentence **from** this synthetic translation

Unsupervised NMT [Artetxe et al., 2017]

- During training, we alternate between denoising and backtranslation training objectives from mini-batch to mini-batch

		FR-EN	EN-FR	DE-EN	EN-DE
Unsupervised	1. Baseline (emb. nearest neighbor)	9.98	6.25	7.07	4.39
	2. Proposed (denoising)	7.28	5.33	3.64	2.40
	3. Proposed (+ backtranslation)	15.56	15.13	10.21	6.55
	4. Proposed (+ BPE)	15.56	14.36	10.16	6.89
Semi-supervised	5. Proposed (full) + 10k parallel	18.57	17.34	11.47	7.86
	6. Proposed (full) + 100k parallel	21.81	21.74	15.24	10.95
Supervised	7. Comparable NMT (10k parallel)	1.88	1.66	1.33	0.82
	8. Comparable NMT (100k parallel)	10.40	9.19	8.11	5.29
	9. Comparable NMT (full parallel)	20.48	19.89	15.04	11.05
	10. GNMT (Wu et al., 2016)	-	38.95	-	24.61

Attention visualisation



Attention in detail

We have encoder hidden states $h_1, \dots, h_N \in \mathbb{R}^h$

On timestep t , we have decoder hidden state $s_t \in \mathbb{R}^h$

We get the attention score \mathbf{e}^t for this step:

$$\mathbf{e}^t = [s_t^T h_1, \dots, s_t^T h_N] \in \mathbb{R}^N$$

We take softmax to get the attention distribution α^t for this step:

$$\alpha^t = \text{softmax}(\mathbf{e}^t) \in \mathbb{R}^N$$

We use alpha to take a weighted sum of the encoder hidden states to get the attention output \mathbf{a}

$$\mathbf{a}_t = \sum_{i=1}^N \alpha_i^t h_i \in \mathbb{R}^h$$

Finally, we concatenate the attention output \mathbf{a}_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model.

$$[\mathbf{a}_t, \mathbf{s}_t] \in \mathbb{R}^{2h}$$

Attention is a powerhouse of Deep Learning technique

More general definition of attention:

Given a set of vectors **values**, and a vector **query**, **attention** is a technique to compute a weighted sum of the values dependent on the query.

Intuition:

- The weighted sum is a **selective summary** of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a fixed-size representation of an arbitrary set of representations (the values), dependent on some other representation (the query)

Upshot:

- Attention has become the powerful, flexible, general way of “pointer and memory” manipulation in deep learning models

There are several attention variants

We have some **values** $h_1, \dots, h_N \in \mathbb{R}^h$ and a **query** $s_t \in \mathbb{R}^h$

Attention always involves:

1. Computing the attention scores
2. Taking softmax to get attention distribution
3. Using attention distribution to take weighted sum of values
4. Thus obtaining the attention output **a** (called context vector)

Different variants

Basic dot-product attention:

$$\mathbf{e}_i = \mathbf{s}^T \mathbf{h}_i \in \mathbb{R}$$

Different variants

Basic dot-product attention:

$$\mathbf{e}_i = \mathbf{s}^T \mathbf{h}_i \in \mathbb{R}$$

Multiplicative attention:

$$\mathbf{e}_i = \mathbf{s}^T \mathbf{W} \mathbf{h}_i \in \mathbb{R}$$

Different variants

Basic dot-product attention:

$$\mathbf{e}_i = \mathbf{s}^T \mathbf{h}_i \in \mathbb{R}$$

Multiplicative attention:

$$\mathbf{e}_i = \mathbf{s}^T \mathbf{W} \mathbf{h}_i \in \mathbb{R}$$

Reduced rank multiplicative attention (for low rank matrices \mathbf{U} and \mathbf{V}):

$$\mathbf{e}_i = \mathbf{s}^T (\mathbf{U}^T \mathbf{V}) \mathbf{h}_i \in \mathbb{R}$$

Different variants

Basic dot-product attention:

$$\mathbf{e}_i = \mathbf{s}^T \mathbf{h}_i \in \mathbb{R}$$

Multiplicative attention:

$$\mathbf{e}_i = \mathbf{s}^T \mathbf{W} \mathbf{h}_i \in \mathbb{R}$$

Reduced rank multiplicative attention (for low rank matrices \mathbf{U} and \mathbf{V}):

$$\mathbf{e}_i = \mathbf{s}^T (\mathbf{U}^T \mathbf{V}) \mathbf{h}_i \in \mathbb{R}$$

Additive attention:

$$\mathbf{e}_i = \mathbf{v}^T \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}) \in \mathbb{R}$$

[Britz et al., 2017]

Attention	newstest2013	Params
Mul-128	22.03 ± 0.08 (22.14)	65.73M
Mul-256	22.33 ± 0.28 (22.64)	65.93M
Mul-512	21.78 ± 0.05 (21.83)	66.32M
Mul-1024	18.22 ± 0.03 (18.26)	67.11M
Add-128	22.23 ± 0.11 (22.38)	65.73M
Add-256	22.33 ± 0.04 (22.39)	65.93M
Add-512	22.47 ± 0.27 (22.79)	66.33M
Add-1028	22.10 ± 0.18 (22.36)	67.11M
None-State	9.98 ± 0.28 (10.25)	64.23M
None-Input	11.57 ± 0.30 (11.85)	64.49M

Table 5: BLEU scores on newstest2013, varying the type of attention mechanism.

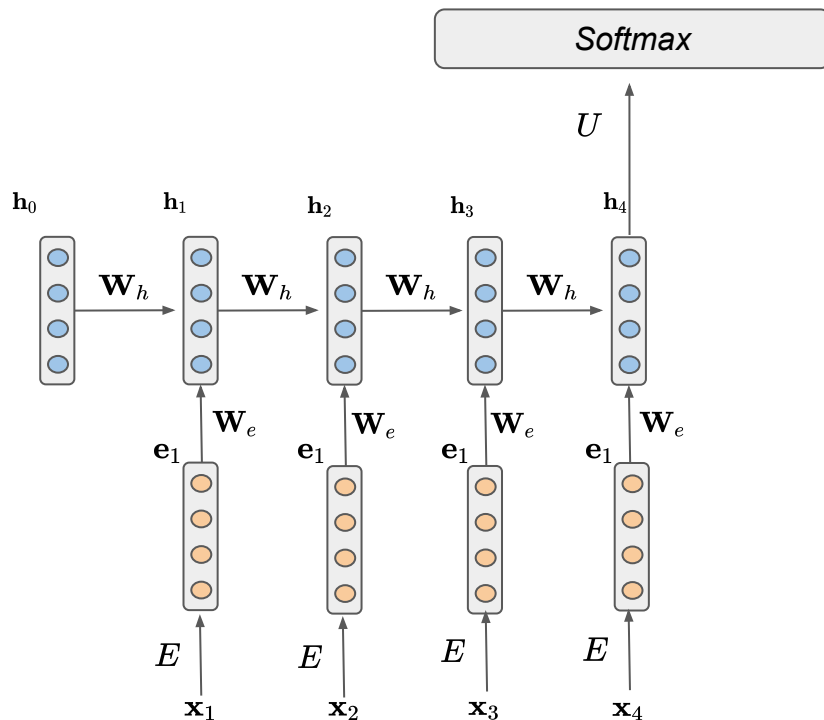
Improvements to RNN

Deep Natural Language Processing Class, 2022
Paweł Budzianowski

Discovering building blocks

1. Meaning
2. Word Vectors
3. New NLP task - language modelling
4. New family of ML models - recurrent neural networks
5. Seq2Seq
6. Attention
7. LSTMs (today)
8. Bidirectional RNNs and Multilayer RNNs (today)

RNN for language modelling



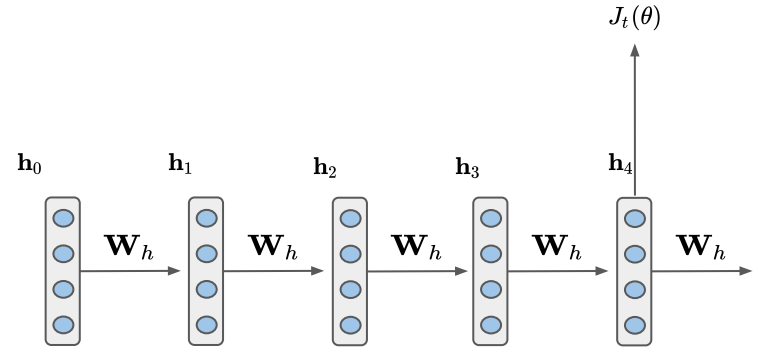
$$y_t = \text{softmax}(U h_t + b_2)$$

$$h_t = \sigma(W_h h_{t-1} + W_e e_t + b_1)$$

$$e_t = E x_t$$

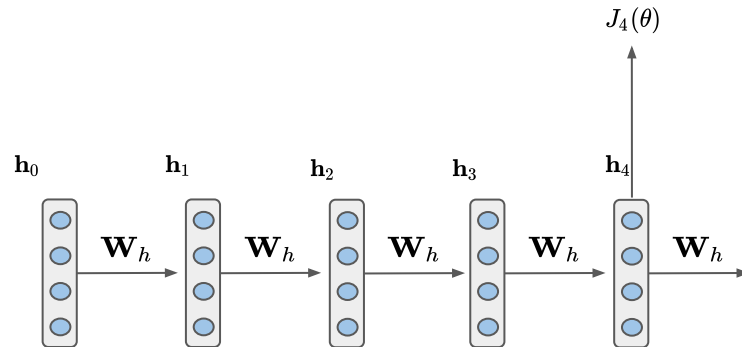
odnóża wylazł z wykrotu

Backpropagation for RNNs



Question: what's the derivative of $J_4(\theta)$ w.r.t \mathbf{h}_1 ?

Backpropagation for RNNs



Question: what's the derivative of $J_4(\theta)$ w.r.t \mathbf{h}_1 ?

Answer: the gradient w.r.t. a repeated weight is the sum of the gradient w.r.t. each time it appears:

$$\frac{\partial \mathbf{J}}{\partial \mathbf{h}_1} = \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \times \frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2} \times \frac{\partial \mathbf{h}_4}{\partial \mathbf{h}_3} \times \frac{\partial \mathbf{J}}{\partial \mathbf{h}_4}$$

Vanishing gradient problem

- Hidden state is: $\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + b_1)$

Vanishing gradient problem

- Hidden state is: $\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + b_1)$
- The general gradient is: $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \text{diag}(\sigma'(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + b_1)) \mathbf{W}_h$

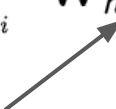
Vanishing gradient problem

- Hidden state is: $\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}_1)$
- The general gradient is: $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \text{diag}(\sigma'(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}_1)) \mathbf{W}_h$
- Consider the gradient of the loss J on step i with respect to the hidden state h on some previous step j :

$$\begin{aligned} \frac{\partial J_i(\theta)}{\partial \mathbf{h}_j} &= \frac{\partial J_i(\theta)}{\partial \mathbf{h}_i} \prod_{j < t \leq i} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \\ &= \frac{\partial J_i(\theta)}{\partial \mathbf{h}_i} \mathbf{W}_{h(i-j)} \prod_{j < t \leq i} \text{diag}(\sigma'(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}_1)) \end{aligned}$$

Vanishing gradient problem

- Hidden state is: $\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}_1)$
- The general gradient is: $\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} = \text{diag}(\sigma'(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}_1)) \mathbf{W}_h$
- Consider the gradient of the loss J on step i with respect to the hidden state h on some previous step j :

$$\begin{aligned} \frac{\partial J_i(\theta)}{\partial \mathbf{h}_j} &= \frac{\partial J_i(\theta)}{\partial \mathbf{h}_i} \prod_{j < t \leq i} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \\ &= \frac{\partial J_i(\theta)}{\partial \mathbf{h}_i} \mathbf{W}_{h(i-j)} \prod_{j < t \leq i} \text{diag}(\sigma'(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}_1)) \end{aligned}$$


If \mathbf{W}_h is small then this term gets vanishingly small as i and j gets apart

Vanishing gradient problem

- L2 norm property gives us:

$$\left\| \frac{\partial \mathbf{J}_i(\theta)}{\partial \mathbf{h}_j} \right\| \leq \left\| \frac{\partial \mathbf{J}_i(\theta)}{\partial \mathbf{h}_i} \right\| \left\| \mathbf{W}_{h_{(i-j)}} \right\| \prod_{j < t \leq i} \left\| \text{diag}(\sigma'(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}_1)) \right\|$$

Vanishing gradient problem

- L2 norm property gives us:

$$\left\| \frac{\partial \mathbf{J}_i(\theta)}{\partial \mathbf{h}_j} \right\| \leq \left\| \frac{\partial \mathbf{J}_i(\theta)}{\partial \mathbf{h}_i} \right\| \left\| \mathbf{W}_{h(i-j)} \right\| \prod_{j < t \leq i} \left\| \text{diag}(\sigma'(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}_1)) \right\|$$

- It was shown that if the largest eigenvalue of \mathbf{W}_h is less than 1, then the gradient will **shrink exponentially**
 - Here the bound is 1 because we have sigmoid nonlinearity

Vanishing gradient problem

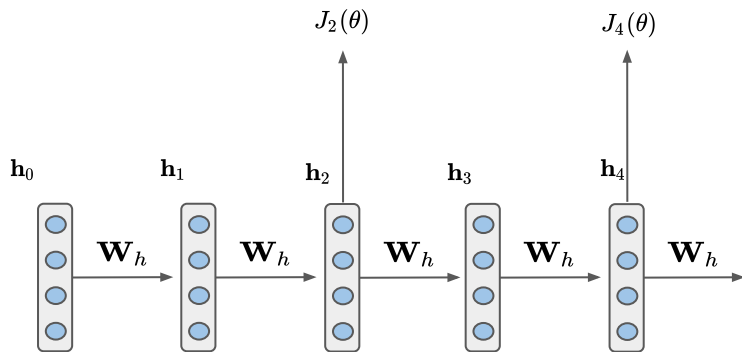
- L2 norm property gives us:

$$\left\| \frac{\partial \mathbf{J}_i(\theta)}{\partial \mathbf{h}_j} \right\| \leq \left\| \frac{\partial \mathbf{J}_i(\theta)}{\partial \mathbf{h}_i} \right\| \left\| \mathbf{W}_{h(i-j)} \right\| \prod_{j < t \leq i} \left\| \text{diag}(\sigma'(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}_1)) \right\|$$

- It was shown that if the largest eigenvalue of \mathbf{W}_h is less than 1, then the gradient will **shrink exponentially**
 - Here the bound is 1 because we have sigmoid nonlinearity
- If the largest eigenvalue > 1 we are bound to **exploding gradients**

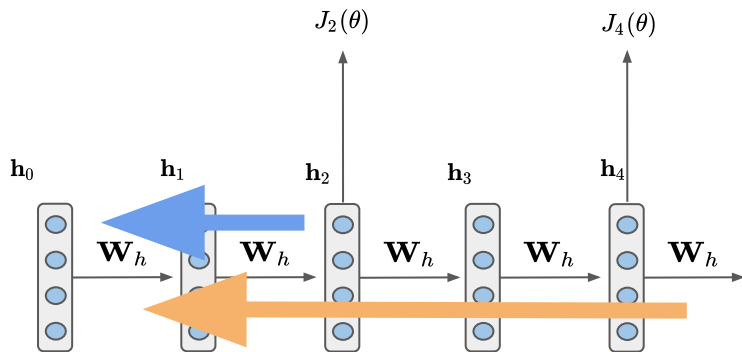
Why is vanishing gradient a problem

- Gradient signal from far away is lost because it's much smaller than the signal from closer steps
- Near effects are more important, not long-term effects.



Why is vanishing gradient a problem

- Gradient signal from far away is lost because it's much smaller than the signal from closer steps
- Near effects are more important, not long-term effects.



Why is vanishing gradient a problem

- Another explanation: gradient can be viewed as a measure of the effect of the past on the future
- If the gradient becomes vanishingly small over longer distances, then we can't tell whether:
 - 1) There's no dependency between step t and $t+n$ in the data
 - 2) We have wrong parameters to capture the true dependency between t and $t+n$

Effect of vanishing gradient on RNN-LM

LM task - *“O, w jednym zdaniu może się dużo zmieścić. Może się wszystko zmieścić. Może się całe życie zmieścić. Zdanie jest wymiarem świata, powiedział filozof. Tak, on. Czasem sobie myślę, czy nie dlatego tyle słów musimy przez życie powiedzieć, aby się mogło z nich wytopić to jedno _____”* (Wiesław Myśliwski, Traktat o łuskaniu fasoli)

The learning signal needs to learn the dependency between the last step and 27th before!

But if gradient is small the model **can't learn this dependency** so the model is unable to predict long-term dependencies.

Effect of vanishing gradient on RNN-LM

- The task: The writer of the books _____ (**is** or **are**)?
- Correct answer the writer of the books is planning a sequel
- Syntactic recency: The **writer** of the books **is**
- Sequential recency: The writer of the **books are**
- RNN-LMs are better at learning from sequential recency than syntactic recency, so they make this type of error more often than we want!

Why is **exploding** gradient a problem?

- If the gradient becomes too big, then the SGD update step becomes too big:

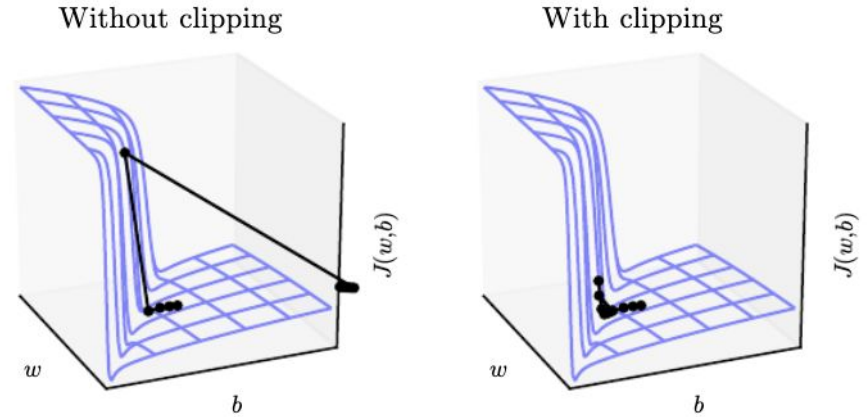
$$\theta_{new} = \theta_{old} - \alpha \nabla_{\theta} J(\theta)$$

- This can cause bad updates: we take too large a step and reach a bad parameter configuration
- In the worst case, this will result in **Inf** or **NaaN** during training.

Gradient clipping

- **Gradient clipping**: if the norm of the gradient is greater than some threshold, scale it down before applying SGD update
- Intuition: take a step in the same direction, but a smaller step

Gradient clipping: example



The cliff is dangerous because it has steep gradient.

How to fix vanishing gradient problem?

- The main problem is that it's too difficult for the RNN to learn to preserve information over many timesteps.
- In a vanilla RNN, the hidden state is constantly being rewritten:

$$\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + b_1)$$

- How about a RNN with separate **memory**?

Long Short-Term Memory (LSTM)

-

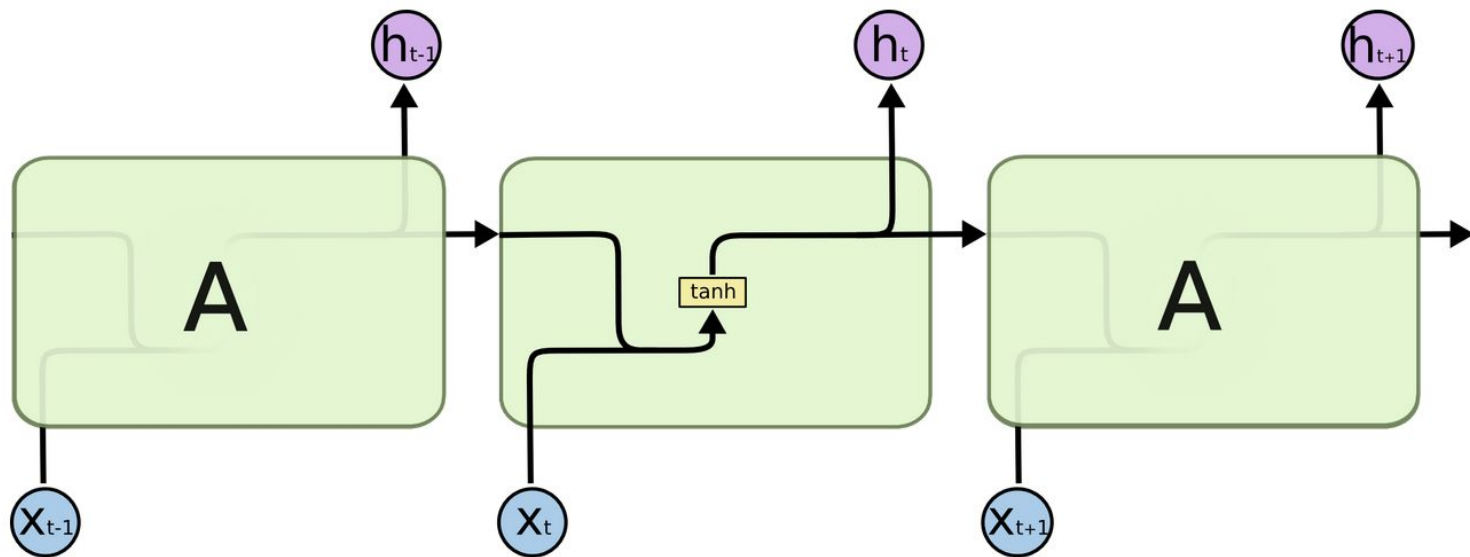
Long Short-Term Memory (LSTM)

- A solution to the vanishing gradients problem [Hochreiter and Schmidhuber, 1997]
- On step t , there is a **hidden state** h_t and a **cell state** c_t
 - both are **vectors** length n
 - the cell stores **long-term information**
 - the LSTM can **erase**, **write** and **read** information from the cell

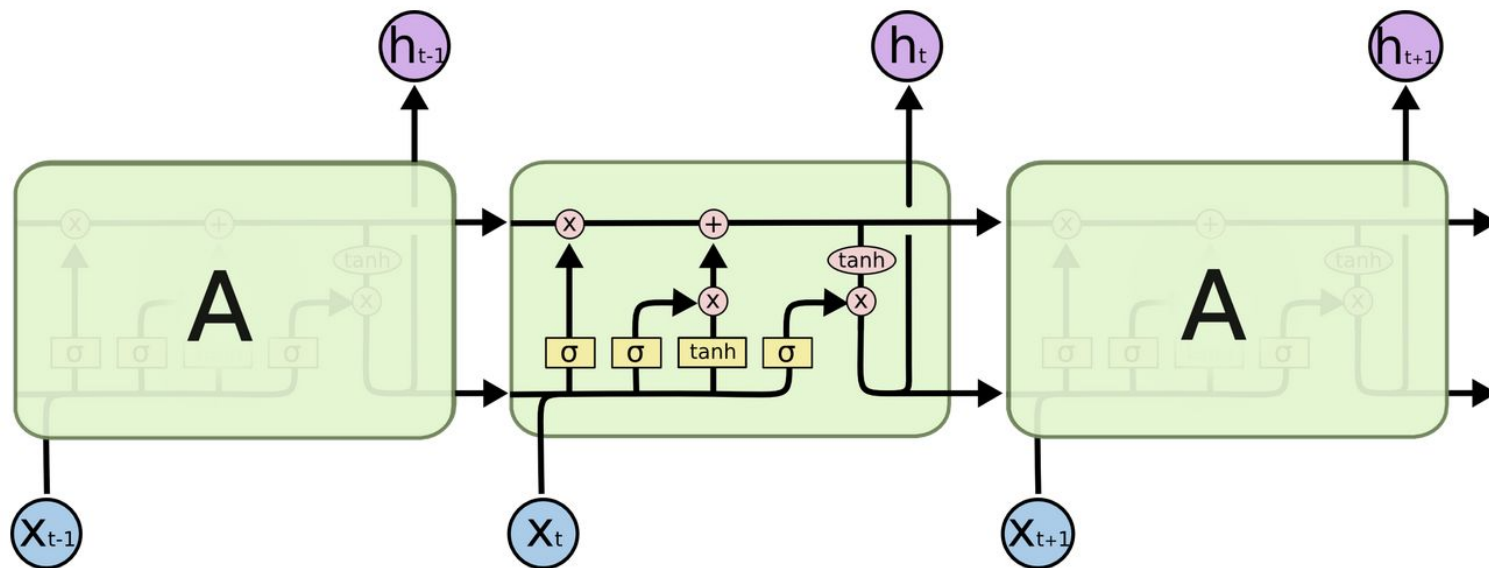
Long Short-Term Memory (LSTM)

- A solution to the vanishing gradients problem [Hochreiter and Schmidhuber, 1997]
- On step t , there is a **hidden state** h_t and a **cell state** c_t
 - both are **vectors** length n
 - the cell stores **long-term information**
 - the LSTM can **erase**, **write** and **read** information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding **gates**:
 - the gates are also vectors length n
 - on each timestep, each element of the gates can be between 0 to 1
 - the gates are **dynamic**: their value is computed based on the current context

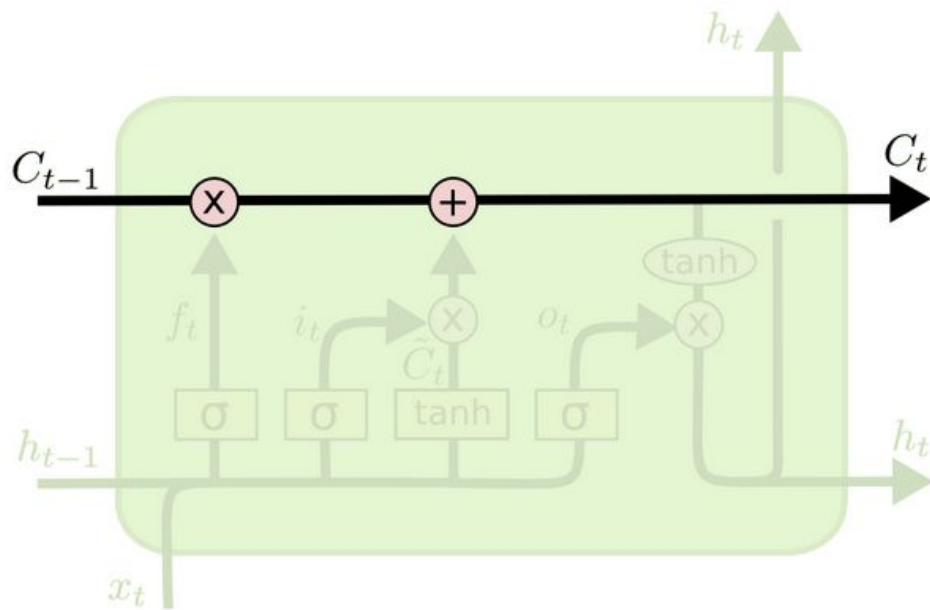
LSTM [Olah, 2015]



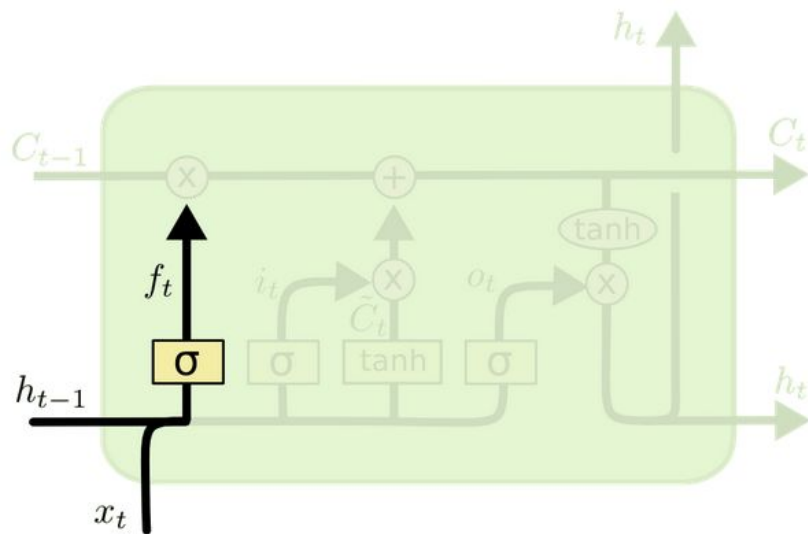
LSTM [Olah, 2015]



Cell state

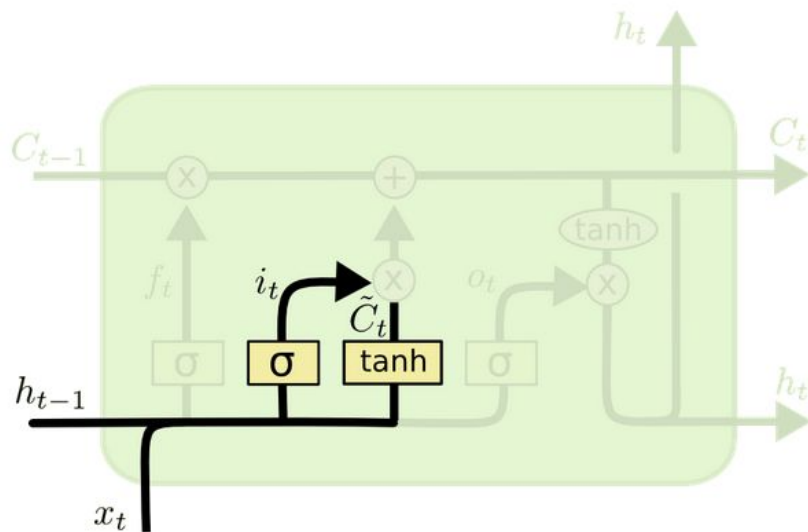


Forget gate



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

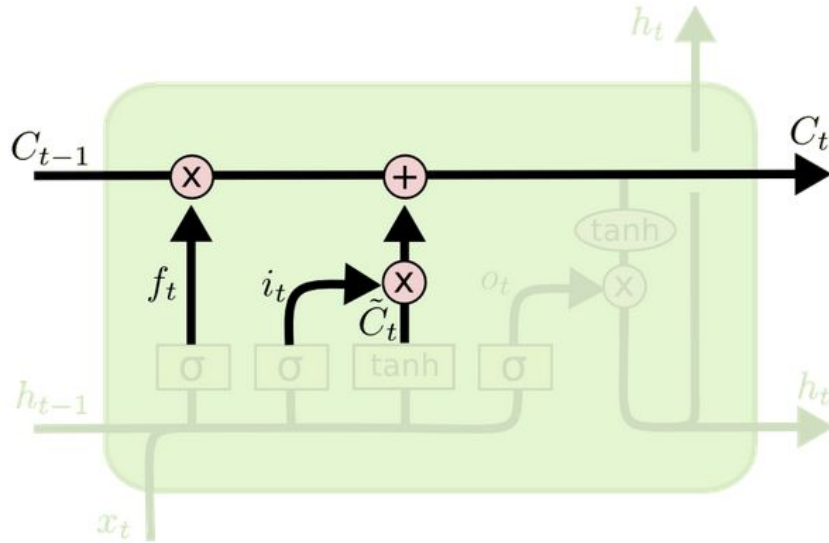
Input gate



$$i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)$$

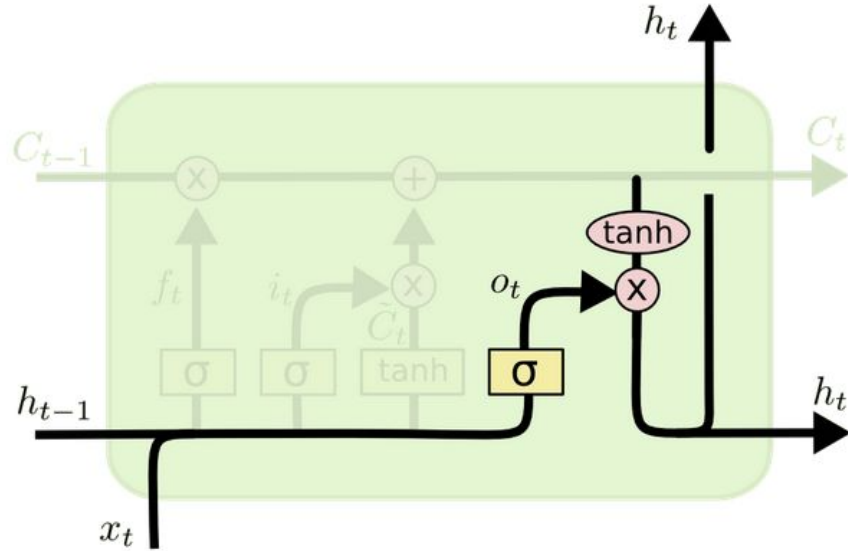
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Cell update



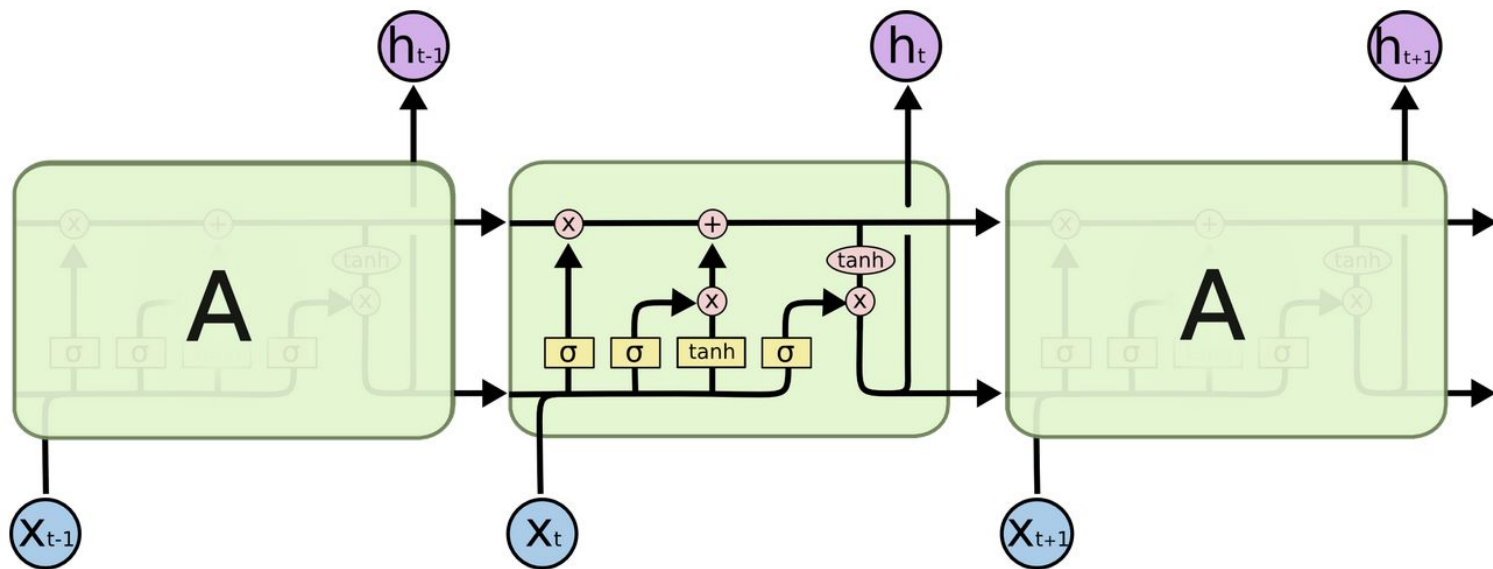
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Next output



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

LSTM [Olah, 2015]



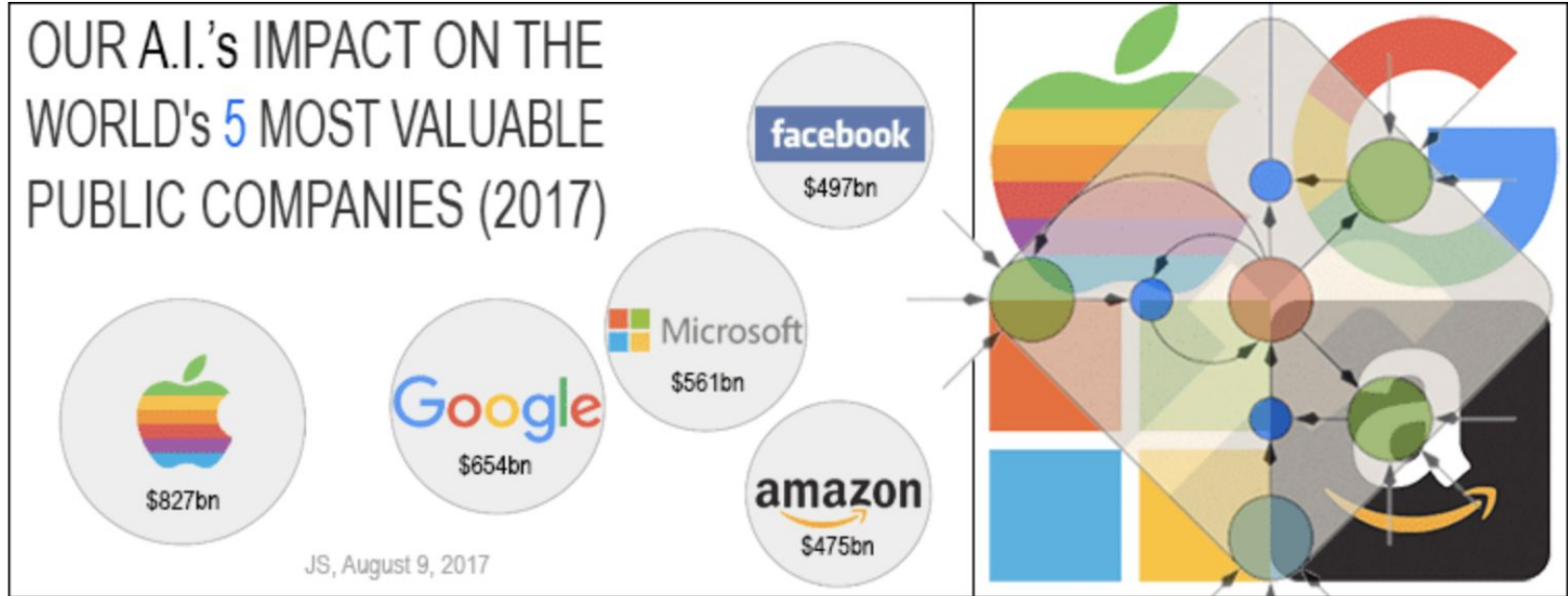
How does LSTM solve our issue

- The LSTM architecture makes it **easier** for the RNN to preserve information **over many timesteps**
- LSMT doesn't guarantee that there is no vanishing/exploding gradient, but it is easier to learn these long-term dependencies

LSTM applications



LSTM applications



Jürgen Schmidhuber (2017, reformatted in 2021)
Pronounce: You_again Shmidhoobuh

AI Blog
@SchmidhuberAI

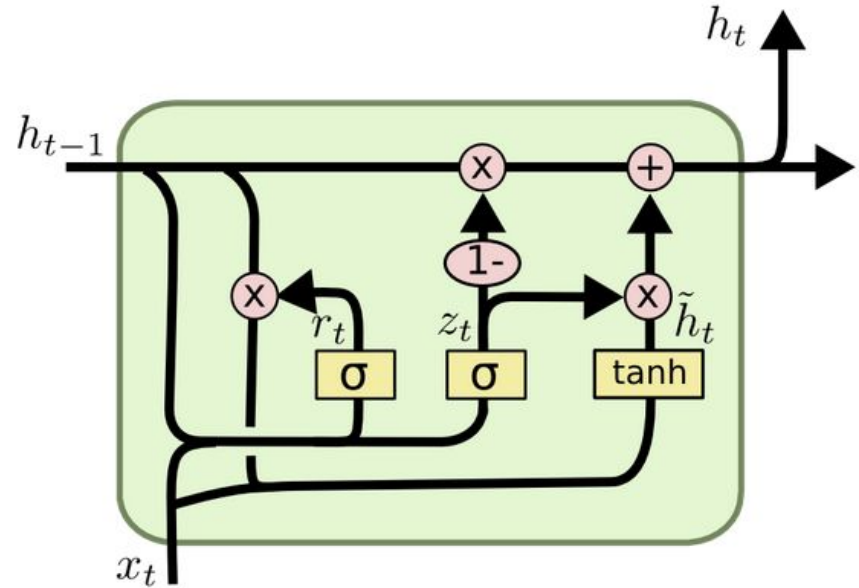
LSTM applications

- LSTM became a default way to go for text encoding since **2013**:
 - ASR
 - Machine Translation
 - Text prediction
 - And many more
- This situation lasted till **2018** when Transformers started to taking over.

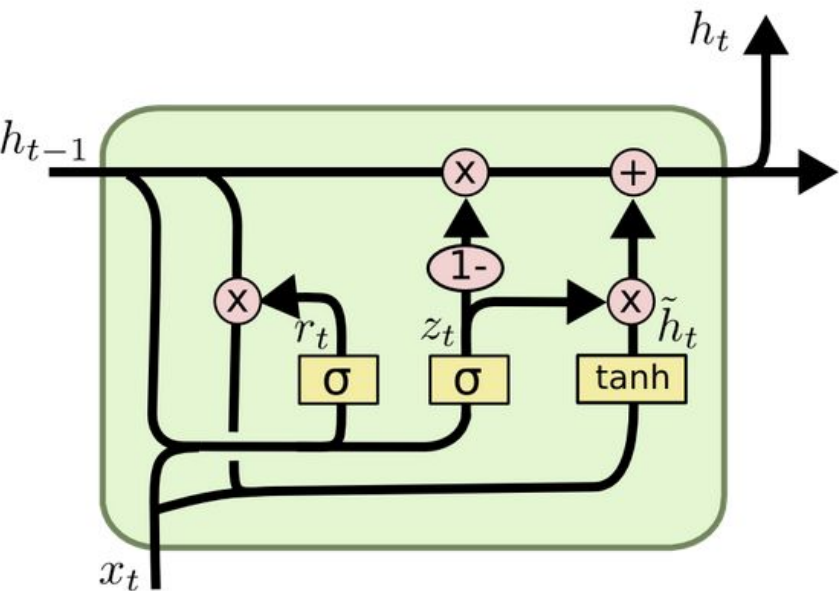
Gated Recurrent Units (GRU) [Cho et al., 2014]

A simpler alternative to LSTM:

- reset gate
- update gate



Gated Recurrent Units (GRU) [Olah, 2015]



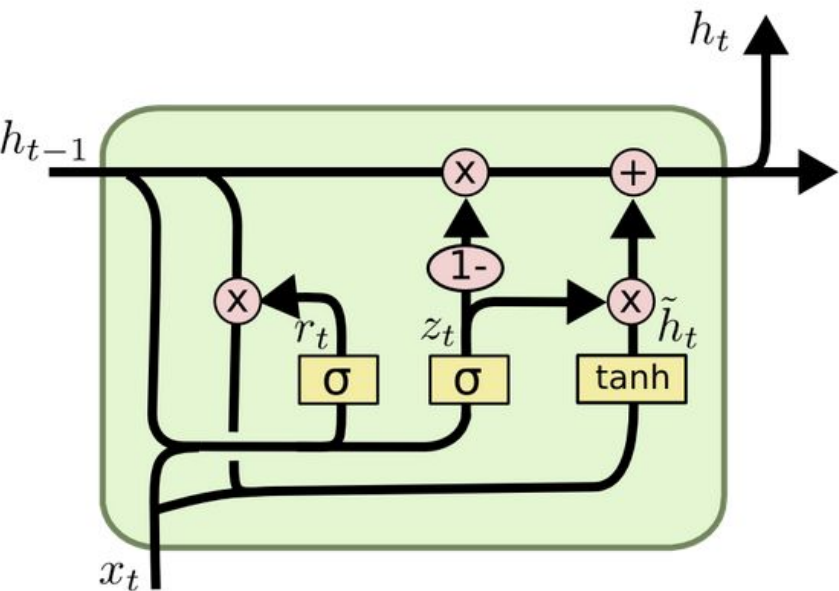
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

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

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

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

Gated Recurrent Units (GRU) [Olah, 2015]



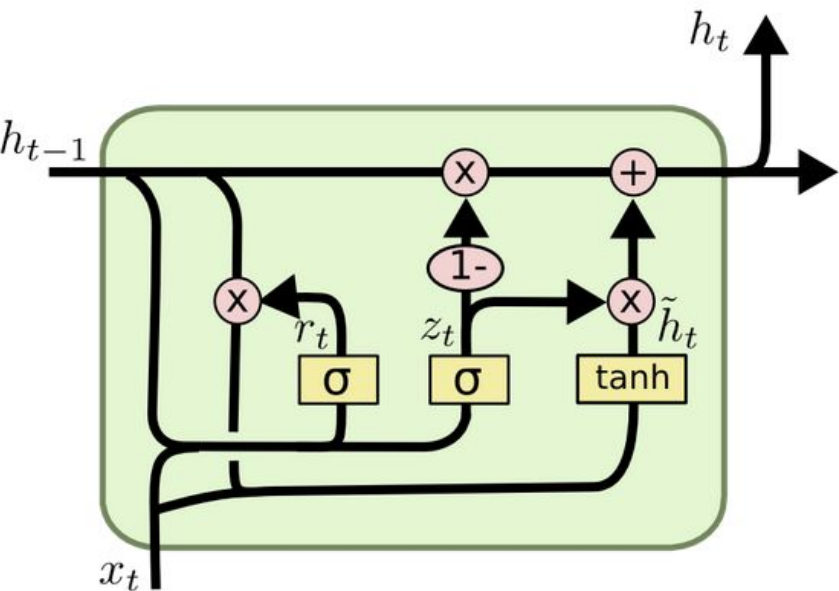
$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

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

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

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

Gated Recurrent Units (GRU) [Olah, 2015]



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

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

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

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

Gated Recurrent Units (GRU)

- The GRU solves the vanishing gradient because it makes **easier** to retain information about long-term by setting **update** gate to 0.
- Again, doesn't solve the vanishing/exploding gradient problem straight away.

LSTM vs GRU

- Researchers have proposed many gated RNN variants, but LSTM and GRU are **the most widely used**
- The biggest difference is that GRU is quicker to compute and has fewer parameters
- There is no conclusive evidence that one consistently performs better than the other
- LSTM can be treated as a default choice and optimize further with GRU once you establish a good performance

GRU/LSTM controversies

LSTM and GRU neural network performance comparison study

taking Yelp review dataset as an example

Shudong Yang*

School of Information and Business Management
Dalian Neusoft University of Information
Dalian, China

* Corresponding author: yangshudong@neusoft.edu.cn

Xueying Yu

School of Information and Business Management
Dalian Neusoft University of Information
Dalian, China

Ying Zhou

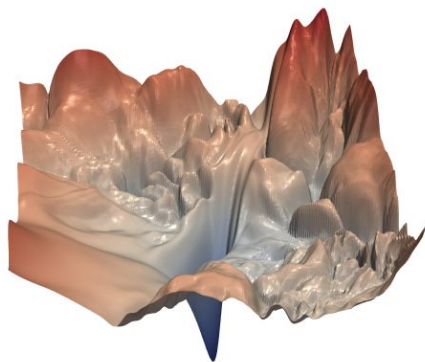
School of Information and Business Management
Dalian Neusoft University of Information
Dalian, China

Is vanishing/exploding gradient just a RNN problem?

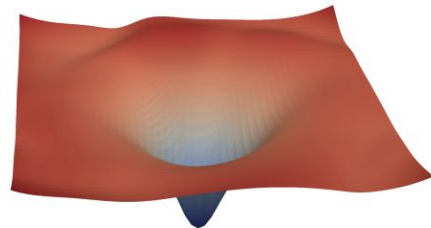
- It can be a problem for all neural architectures, especially deep ones
- ReLU, tanh
- Residual layers (Res-nets)

Is vanishing/exploding gradient just a RNN problem?

- It can be a problem for all neural architectures, especially deep ones
- ReLU, tanh
- Residual layers (Res-nets)



(a) without skip connections



(b) with skip connections

Is vanishing/exploding gradient just a RNN problem?

- It can be a problem for all neural architectures, especially deep ones
- ReLU, tanh
- Residual layers (Res-nets)
- Dense net - connect everything to everything

Is vanishing/exploding gradient just a RNN problem?

- It can be a problem for all neural architectures, especially deep ones
- ReLU, tanh
- Residual layers (Res-nets)
- Dense net - connect everything to everything
- Highway connections - connection controlled by a dynamic gate

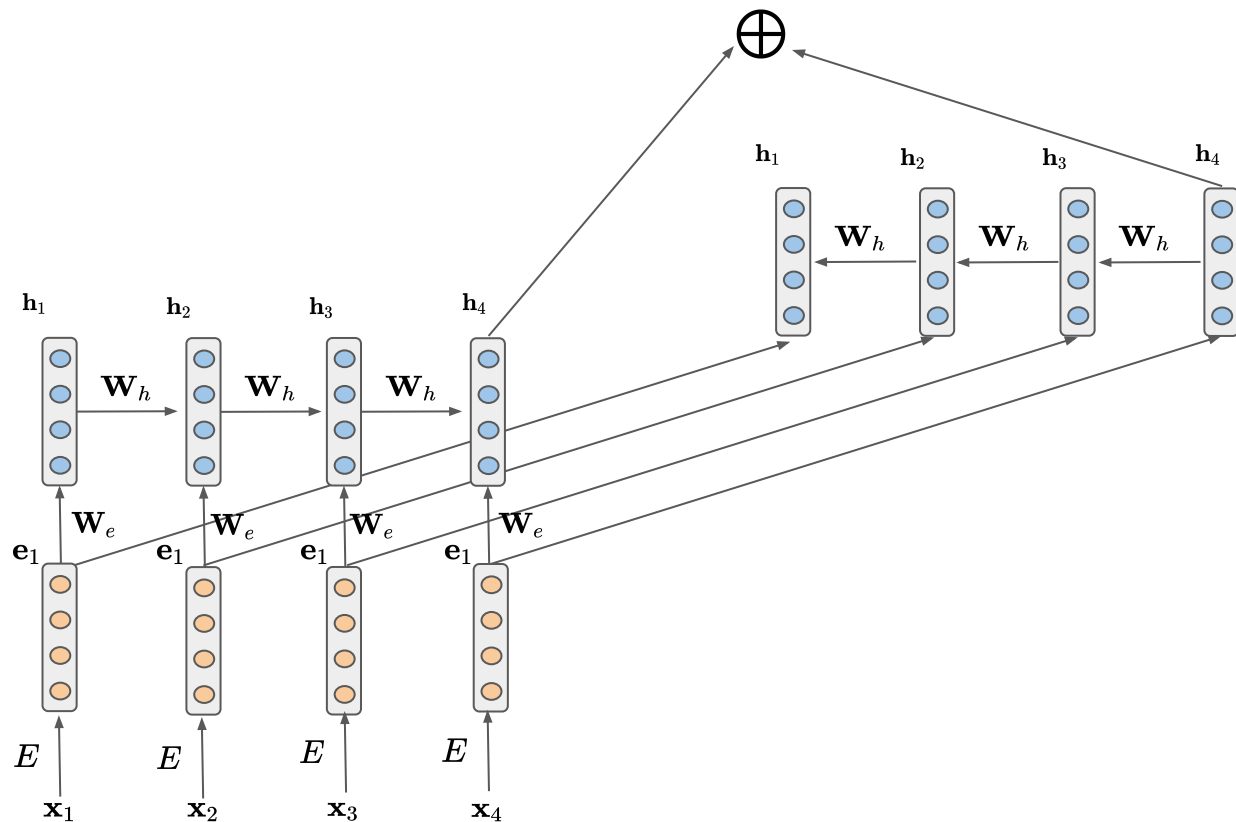
RNN vs General Deep Networks

Although vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the **same weight matrix**

Context matters!

- Example: Stary, to było strasznie dobre!

Bidirectional RNNs



to było strasznie dobre

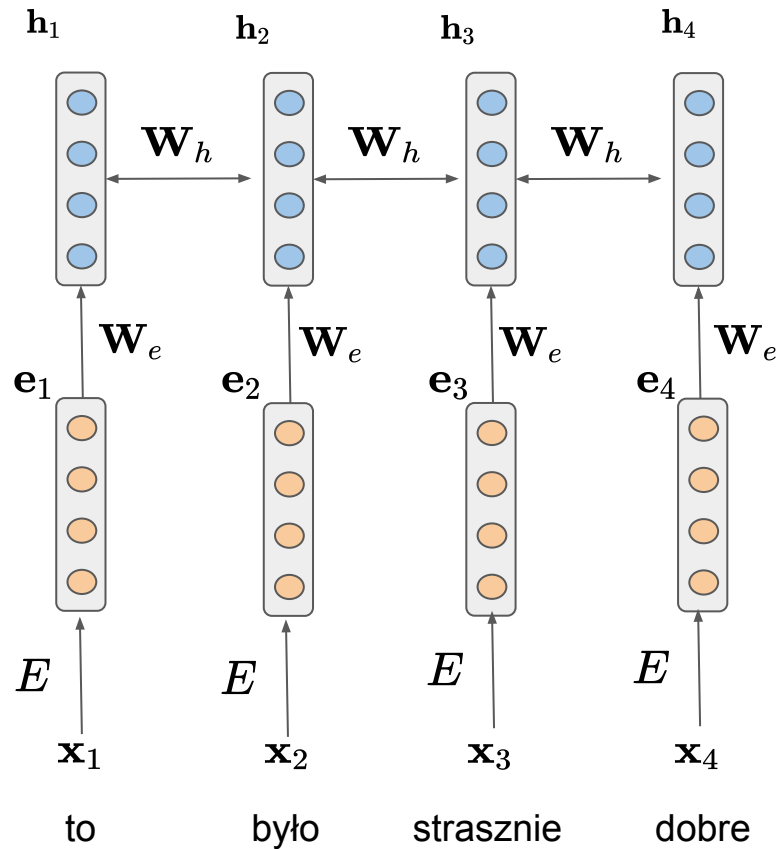
Bidirectional RNNs

$$\overrightarrow{\mathbf{h}}_t = \text{RNN}_{\text{FW}}(\overrightarrow{\mathbf{h}}_{t-1}, \mathbf{x}_t)$$

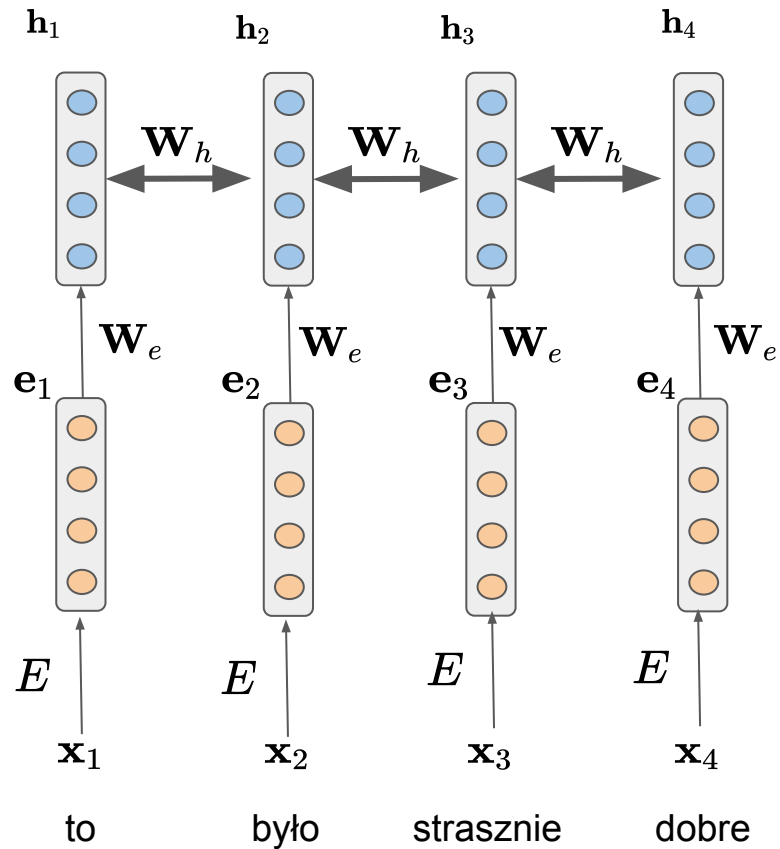
$$\overleftarrow{\mathbf{h}}_t = \text{RNN}_{\text{BW}}(\overleftarrow{\mathbf{h}}_{t+1}, \mathbf{x}_t)$$

$$\mathbf{h}_t = [\overrightarrow{\mathbf{h}}_t, \overleftarrow{\mathbf{h}}_t]$$

Bidirectional RNNs



Bidirectional RNNs



Bidirectional RNNs

- By definition, we have to have an access to the entire sequence for modelling in both directions
- Therefore, we can't use the Bidirectional RNNs for Language Modelling since we would be cheating by looking ahead

Multi-layer RNNs

- RNNs are **already** deep on one dimension
- We can also make them deep in another dimension by **applying multiple RNNs**

Multi-layer RNNs

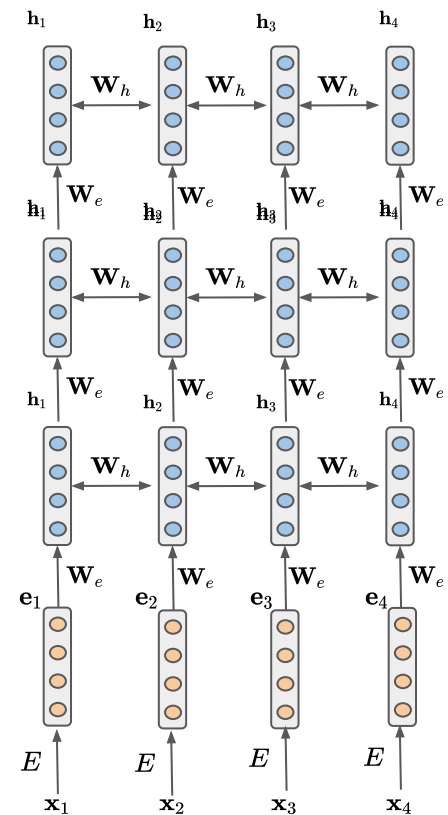
- RNNs are **already** deep on one dimension
- We can also make them deep in another dimension by **applying multiple RNNs**
- This allows the network to compute more complex representations
- They are sometimes called **stacked** RNNs

Multi-layer RNNs

Layer 3

Layer 2

Layer 1



to było strasznie dobre

Multi-layer RNNs graph

- Most best models were deep in 2015-2018 years
- The average models were between 2 to 8 layers
- Adding more layers requires skip-connections or dense layers
- They are also expensive to compute

Literature

1. Pascanau et al., 2013 - <https://arxiv.org/abs/1211.5063>
2. Yang et al., 2016 - <https://ieeexplore.ieee.org/abstract/document/9221727>
3. Hochreiter and Schmidhuber, 1997 - <https://ieeexplore.ieee.org/abstract/document/6795963/>
4. Britz et al., 2017 - <https://arxiv.org/pdf/1703.03906.pdf>