Deep Learning

Lecture 7: Recurrent neural networks

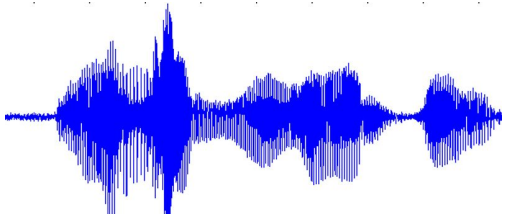
Prof. Stéphane Gaïffas
https://stephanegaiffas.github.io

Today

How to use representation learning with sequential data ?

- Embeddings
- Temporal convolutions
- Recurrent neural networks
- Applications
- Beyond sequences

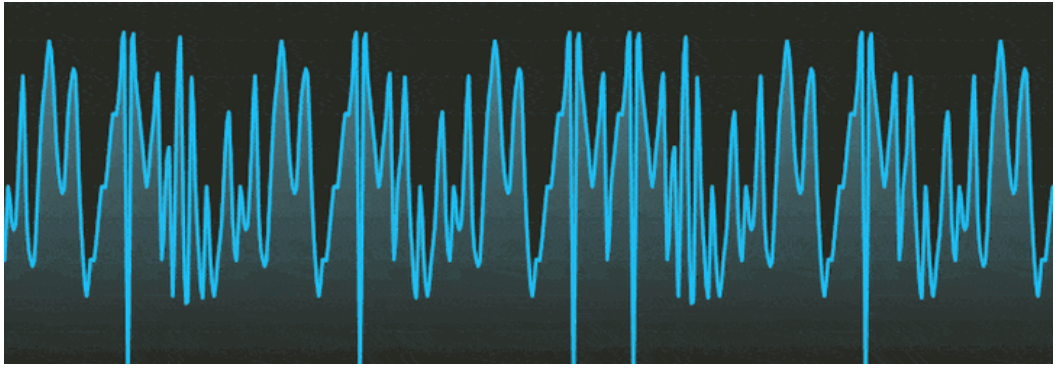
Examples of sequences



Audio signals

Lacus laoreet non curabitur gravid
Faucibus interdum posuere lorem i
Aliquam faucibus purus in massa t
Auctor urna nunc id cursus metus.
Velit egestas dui id ornare arcu odi
Et tortor consequat id porta nibh v
Id diam maecenas ultricies mi eget
Fermentum odio eu feugiat pretiun
Tincidunt nunc pulvinar sapien et
Semper quis lectus nulla at volutpa
Elementum tempus egestas sed sed
Mauris pellentesque pulvinar pelle
Viverra maecenas accumsan lacus

Text

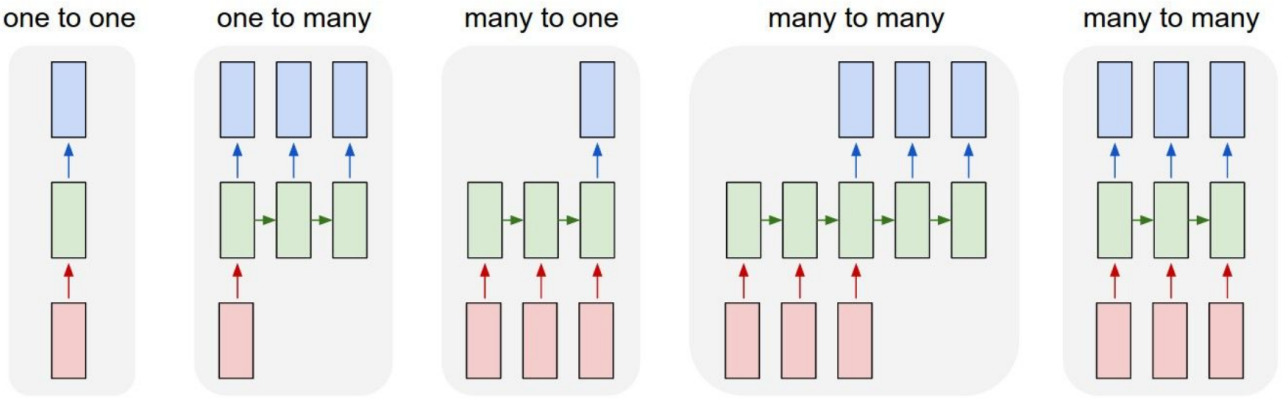


Time series

Real-world problems with a sequence structure

- Sequence classification:
  - sentiment analysis
  - activity/action recognition
  - DNA sequence classification
  - action selection
- Sequence synthesis:
  - text synthesis
  - music synthesis
  - motion synthesis
- Sequence-to-sequence translation:
  - speech recognition
  - text translation
  - part-of-speech tagging

Prediction problems



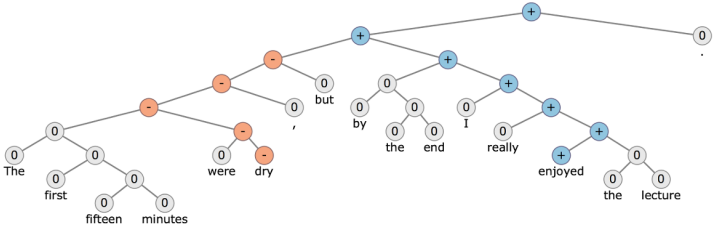
- One to one. Vanilla Neural Networks
- One to many. Image Captioning: image/sequence of words
- Many to one. Sentiment classification: sequence of words/sentiment
- Many to many. Translation: sequence of words/sequence of words
- Many to many. Video classification on frame level: sequence of image/sequence of label

Limitations of (feed-forward) neural networks

- They can't deal with sequential or "temporal" data
- They lack memory
- They have a fixed architecture: fixed input size and fixed output size

Recurrent neural networks are used to overcome successfully these limitations

[Google Translate System - 2016]



[Socher 2015]

Embeddings

Mostly for Natural Language Processing (NLP)

Natural Language Processing

- Sentence/Document level Classification (topic, sentiment)
- Topic modeling (LDA, ...)
- Translation
- Chatbots / dialogue systems / assistants

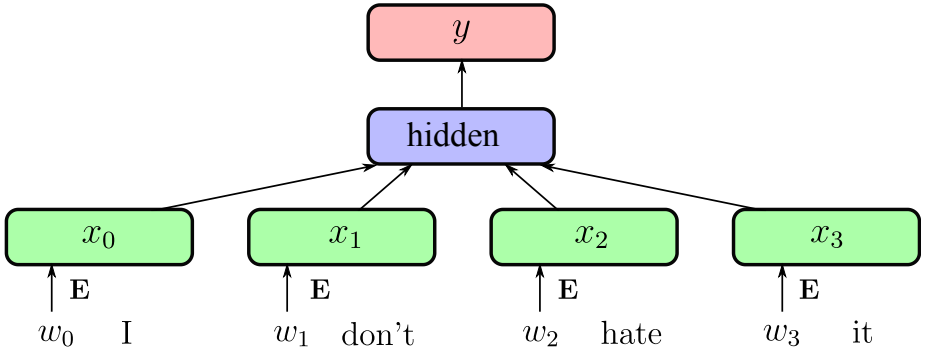
Useful open source projects

- `gensim`
- `spacy`
- `huggingface`

Word representation

- Words are basically indexed (replaced by an integer) and/or represented as 1-hot vectors: "dog" becomes 42
- Possibly very large vocabulary V of possible words
- We can replace words by word-pieces using a tokenizer to end up with 20K tokens
- In nearly all deep learning tasks, words or tokens are replaced by embeddings or embedding vectors
- Each token is replaced by a vector $e \in \mathbb{R}^E$ where E is the dimension of the embeddings, usually between 32 and 512
- Embeddings are learned just as any other weight in the neural network
- Parameter sharing: same embedding each time the same token is seen
- Word / token embeddings are assets for most languages

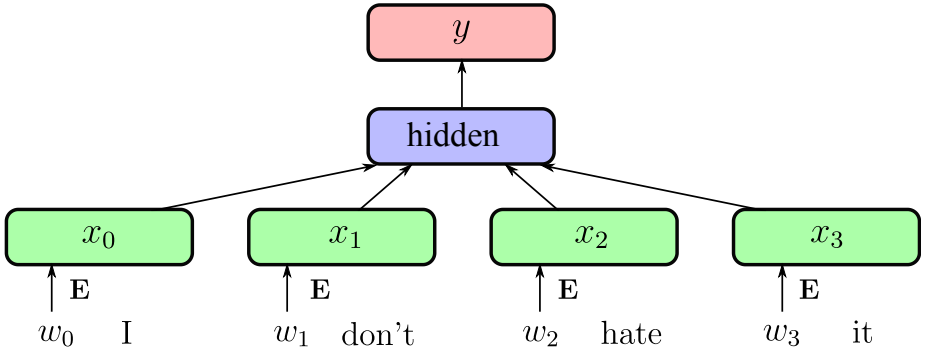
Embeddings for text classification



- First layer: embedding $\mathbf{E}$ of size $|V| \times E$
- Embeddings are averaged across the sequence of words: a single vector $h \in \mathbb{R}^E$ describes the whole sentence
- h is fed to a dense layer that predicts the label, with softmax activation and cross-entropy loss

Joulin, Armand, et al. "Bag of tricks for efficient text classification." FAIR 2016

Embeddings for text classification

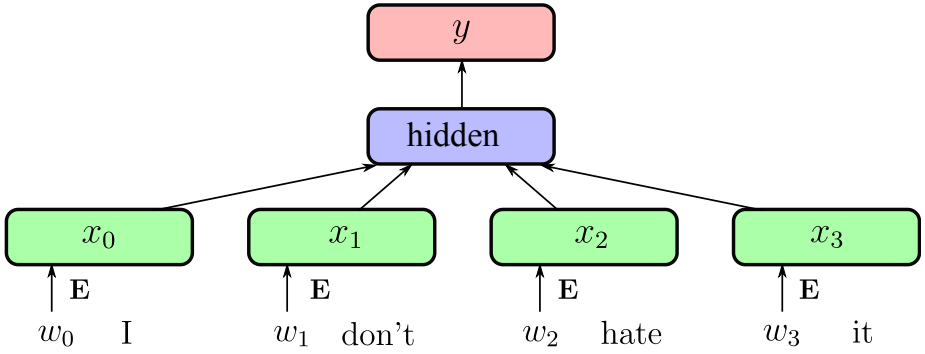


- Input $x \in \mathbb{N}^{\text{length(x)}}$
- $u = \text{Embedding}(x) \in \mathbb{R}^{E \times \text{length(x)}}$
- $v = \text{mean}(u) \in \mathbb{R}^{E}$
- $y = \text{softmax}(Wv + b) \in \Delta_K$ where $\Delta_K \subset \mathbb{R}^K$ is the simplex

---

Joulin, Armand, et al. "Bag of tricks for efficient text classification." FAIR 2016

Embeddings for text classification



- Very efficient (speed and accuracy) on large datasets
- State-of-the-art (or close to) on several classification, when adding bigrams/trigrams
- Little gains from depth

Joulin, Armand, et al. "Bag of tricks for efficient text classification." FAIR 2016

Transfer learning for text

- Can we use transfer learning with word embeddings ? Similar to images: can we have word representations that are generic enough to transfer from one task to another?
- Unsupervised or self-supervised learning of word representations
- Unlabelled text data is almost infinite: Wikipedia dumps, Project Gutenberg, Social Networks, etc.

Idea: use self-supervised training

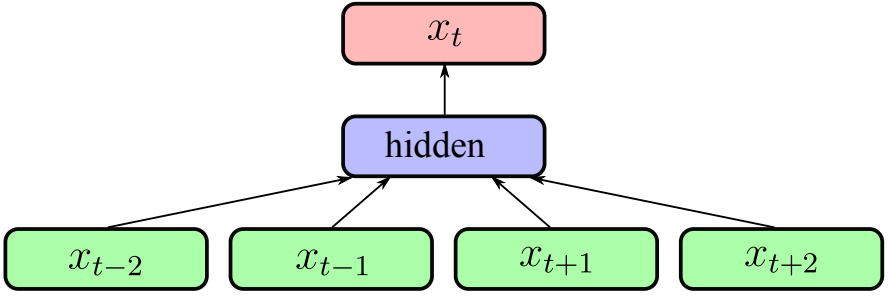
- Distributional Hypothesis (Harris, 1954): "words are characterized by the company they keep"
- Learn word embeddings by predicting word contexts. Given a word e.g. "`dog`" and any other word $w \in V$, learn to predict the probability of $\mathbb{P}[w|\text{dog}]$ that w occurs in the context of "`dog`"
- Unsupervised or self-supervised: no need for class labels. Self-supervision comes from context. Tons of text required to cover rate words correctly

Word2Vec: CBoW

- CBoW: representing the context as a Continuous Bag-of-Word
- Self-supervision from large unlabeled corpus of text: slide over an anchor word and its context:

```
the carrot is a root vegetable, usually orang
```

- Similar to text classification with $|V|$ classes



- Problem: huge number of classes !

Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." NIPS 2013

Word2Vec: CBoW

- Problem: huge number of classes ! Softmax involves a sum over the whole vocabulary V at each gradient step : Computationally intractable

Trick: negative sampling

- Sample negative words at random instead of computing the full softmax
- Use say $k = 5$ negative words sampled at random instead. Not accurate enough to estimate accurately

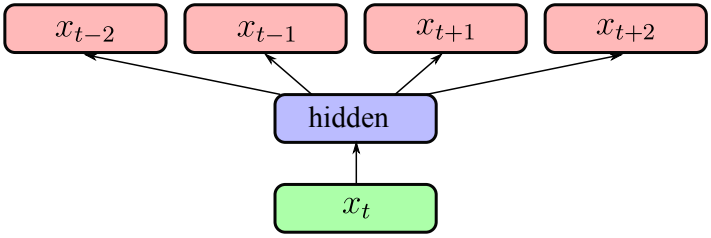
$$\mathbb{P}\left[x_t \middle| x_{t-2}, x_{t-1}, x_{t+1}, x_{t+2}\right]$$

- But it's a good enough approximation to train useful word embedding parameters

---

Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." NIPS 2013
http://sebastianruder.com/word-embeddings-softmax/index.html

Word2Vec: Skip Gram



- Given the central word, predict occurence of other words in its context.
- Widely used in practice
- Again Negative Sampling is used as a cheaper alternative to full softmax

Evaluation and Related methods

Always difficult to evaluate unsupervised tasks

- Other popular method: GloVe (Socher et al.)
- http://nlp.stanford.edu/projects/glove/

Word2Vec

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Colobert et al. 2011, Mikolov, et al. 2013

Word2Vec

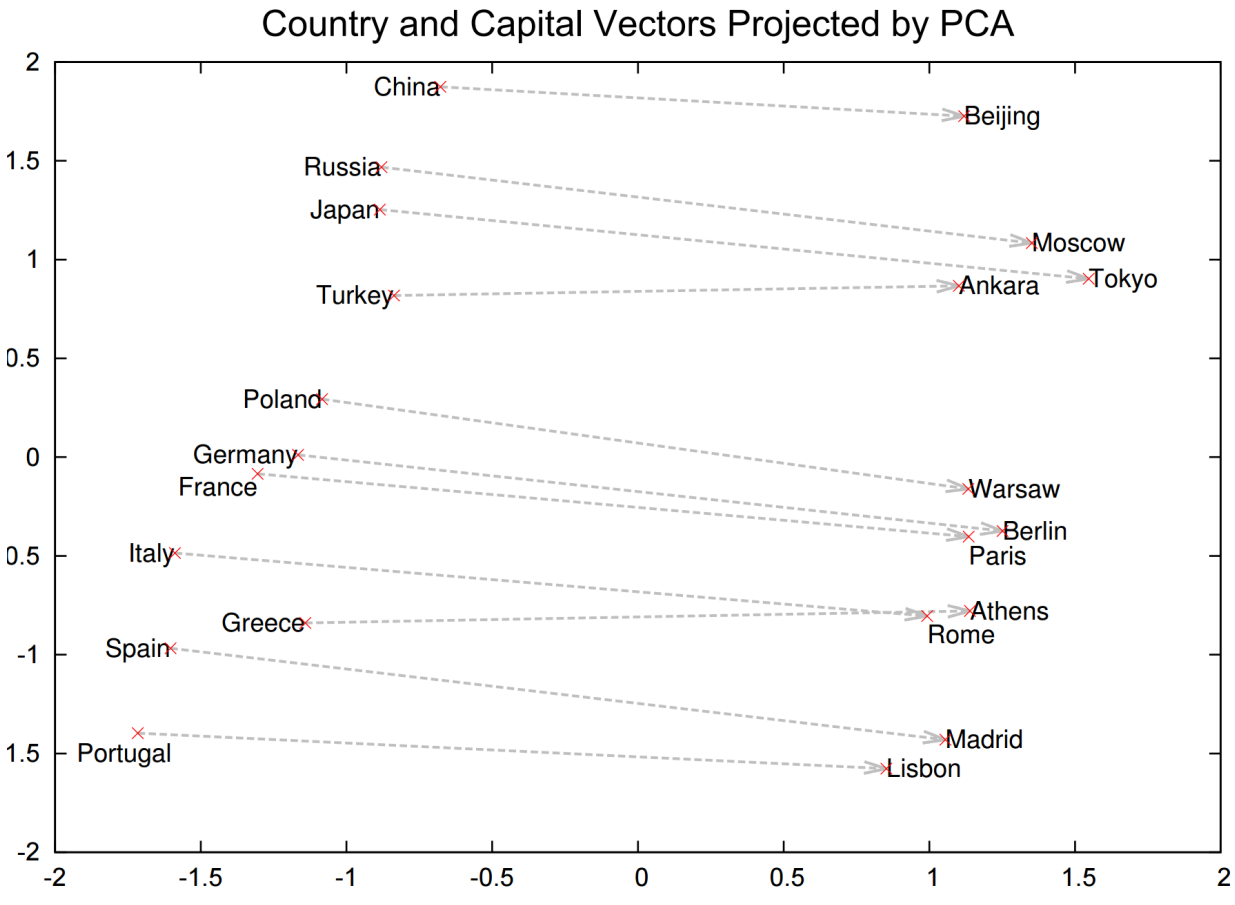
FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Compositionality

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

Colobert et al. 2011, Mikolov, et al. 2013

Word analogies



Take away on embeddings

For text applications, inputs of neural networks are embeddings

- Little training data and large vocabulary not well covered by training data: use transfer learning with pre-trained embeddings
- Large training data with labels: learn directly task-specific embedding in supervised mode
- These methods use Bag-of-Words (BoW): they ignore the order in word sequences
- Depth and non-linear activations on hidden layers are not that useful for BoW text classification

Language models

Language models

- Assign a probability to a sequence of words
- Such that plausible sequences have higher probabilities:

$$\mathbb{P}[\texttt{"I like cats"}] > \mathbb{P}[\texttt{"I table cats"}]$$

$$\mathbb{P}[\texttt{"I like cats"}] > \mathbb{P}[\texttt{"like I cats"}]$$

- Order is important $\neq$ bag-of-Words representations from above

Language models

- Assign a probability to a sequence of words
- Such that plausible sequences have higher probabilities:

$$\mathbb{P}[\texttt{"I like cats"}] > \mathbb{P}[\texttt{"I table cats"}]$$

$$\mathbb{P}[\texttt{"I like cats"}] > \mathbb{P}[\texttt{"like I cats"}]$$

- Order is important $\neq$ bag-of-Words representations from above

Auto-regressive **sequence modeling**

$$\mathbb{P}_\theta[w_0]$$

$\mathbb{P}_\theta$ is parametrized by a neural network

Language models

- Assign a probability to a sequence of words
- Such that plausible sequences have higher probabilities:

$$\mathbb{P}[\texttt{"I like cats"}] > \mathbb{P}[\texttt{"I table cats"}]$$

$$\mathbb{P}[\texttt{"I like cats"}] > \mathbb{P}[\texttt{"like I cats"}]$$

- Order is important $\neq$ bag-of-Words representations from above

Auto-regressive **sequence modeling**

$$\mathbb{P}_\theta[w_0] \times \mathbb{P}_\theta[w_1|w_0]$$

$\mathbb{P}_\theta$ is parametrized by a neural network

Language models

- Assign a probability to a sequence of words
- Such that plausible sequences have higher probabilities:

$$\mathbb{P}[\texttt{"I like cats"}] > \mathbb{P}[\texttt{"I table cats"}]$$

$$\mathbb{P}[\texttt{"I like cats"}] > \mathbb{P}[\texttt{"like I cats"}]$$

- Order is important $\neq$ bag-of-Words representations from above

Auto-regressive **sequence modeling**

$$\mathbb{P}_\theta[w_0] \times \mathbb{P}_\theta[w_1|w_0] \times \cdots \times \mathbb{P}_\theta[w_n|w_{n-1}, w_{n-2}, \ldots, w_0]$$

$\mathbb{P}_\theta$ is parametrized by a neural network

Language models

- Assign a probability to a sequence of words
- Such that plausible sequences have higher probabilities:

$$\mathbb{P}[\texttt{"I like cats"}] > \mathbb{P}[\texttt{"I table cats"}]$$

$$\mathbb{P}[\texttt{"I like cats"}] > \mathbb{P}[\texttt{"like I cats"}]$$

- Order is important $\neq$ bag-of-Words representations from above

Auto-regressive **sequence modeling**

$$\mathbb{P}_\theta[w_0] \times \mathbb{P}_\theta[w_1|w_0] \times \cdots \times \mathbb{P}_\theta[w_n|w_{n-1}, w_{n-2}, \ldots, w_0]$$

$\mathbb{P}_\theta$ is parametrized by a neural network

Hope: internal representation of the model can better capture the meaning of a sequence than a Bag-of-Words

Conditional Language Models

NLP problems expressed as conditional language models

Example 1. Translation problem

- Consider

$$\mathbb{P}[\texttt{target}|\texttt{source}]$$

- Example: **`source`**: `"J'aime les chats"` and **`target`**: `"I love cats"`
- Model the output word by word

$$\mathbb{P}_\theta[w_0|\texttt{source}] \times \mathbb{P}_\theta[w_1|w_0, \texttt{source}] \times \cdots$$

Conditional Language Models

NLP problems expressed as conditional language models

Example 2. Question Answering / Dialogue / Chatbot

- Consider

$$\mathbb{P}[\texttt{answer}|\texttt{question}, \texttt{context}]$$

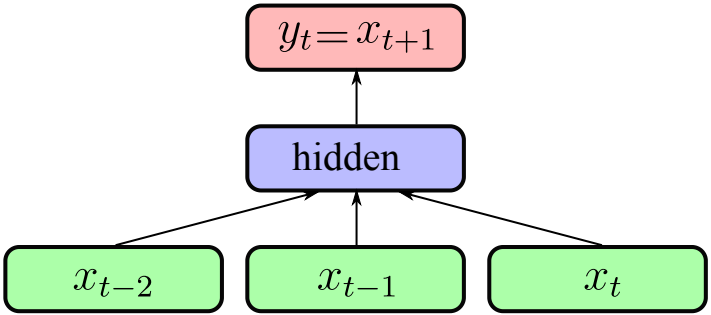
- `context`: "John puts two glasses on the table."; "Bob adds two more glasses."; "Bob leaves the kitchen to play baseball in the garden."
- `question`: "How many glasses are there?"
- `answer`: "There are four glasses."

Example 3. Image captioning

- Consider $\mathbb{P}[\texttt{caption}|\texttt{image}]$
- `image` $=$ a flat representation of the image in $\mathbb{R}^h$ output by a CNN

A simple language model

If we use the same idea as before



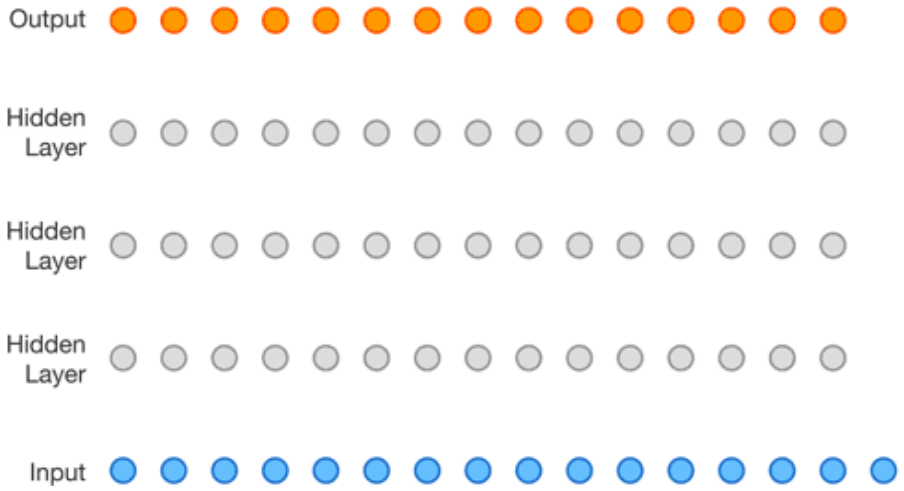
- Fixed context size
- Average embeddings (same as CBoW): no sequence information
- Concatenate embeddings: introduces many parameters
- 1D convolution: larger contexts, limit number of parameters
- Does not take well into account varying sequence sizes and sequence dependencies

Temporal convolutions

Temporal convolutions

The simplest approach to sequence processing is to use temporal convolutional networks (TCNs).

TCNs correspond to standard 1D convolutional networks. They process input sequences as fixed-size vectors of the maximum possible length.



A Oord et al. Wavenet: A generative model for raw audio (2016)

Temporal convolutions

Increasing exponentially the kernel sizes makes the required number of layers grow as $O(\log T)$ of the time window T taken into account.

Dilated convolutions make the model size grow as $O(\log T)$, while the memory footprint and computation are $O(T \log T)$.

Credits: Philippe Remy, keras-tcn, 2018; Francois Fleuret, EE559 Deep Learning, EPFL.

Temporal convolutions

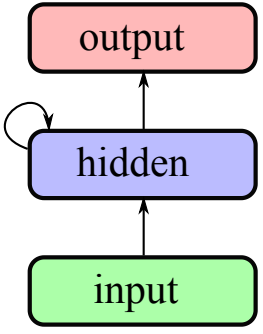
Table 1. Evaluation of TCNs and recurrent architectures on synthetic stress tests, polyphonic music modeling, character-level language modeling, and word-level language modeling. The generic TCN architecture outperforms canonical recurrent networks across a comprehensive suite of tasks and datasets. Current state-of-the-art results are listed in the supplement. ^{h} means that higher is better. $^\ell$ means that lower is better.

Sequence Modeling Task	Model Size ($\approx$)	Models			
		LSTM	GRU	RNN	**TCN**
Seq. MNIST (accuracy^h)	70K	87.2	96.2	21.5	**99.0**
Permuted MNIST (accuracy)	70K	85.7	87.3	25.3	**97.2**
Adding problem T=600 (loss$^\ell$)	70K	0.164	**5.3e-5**	0.177	**5.8e-5**
Copy memory T=1000 (loss)	16K	0.0204	0.0197	0.0202	**3.5e-5**
Music JSB Chorales (loss)	300K	8.45	8.43	8.91	**8.10**
Music Nottingham (loss)	1M	3.29	3.46	4.05	**3.07**
Word-level PTB (perplexity$^\ell$)	13M	**78.93**	92.48	114.50	88.68
Word-level Wiki-103 (perplexity)	-	48.4	-	-	**45.19**
Word-level LAMBADA (perplexity)	-	4186	-	14725	**1279**
Char-level PTB (bpc$^\ell$)	3M	1.36	1.37	1.48	**1.31**
Char-level text8 (bpc)	5M	1.50	1.53	1.69	**1.45**

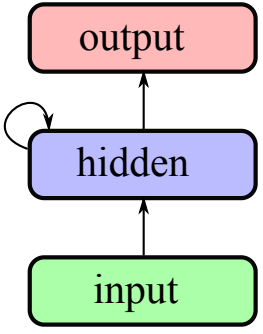
Credits: Bai et al, An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling, 2018.

Recurrent Neural Network

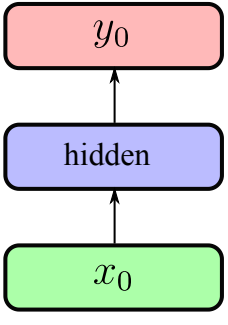
Recurrent Neural Network



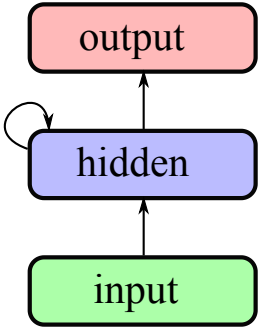
Recurrent Neural Network



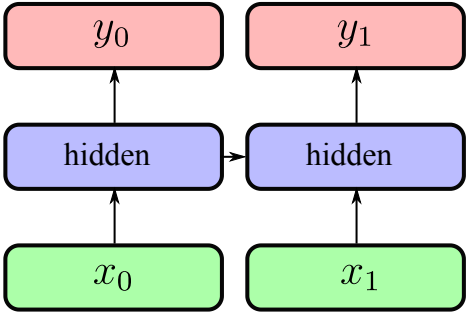
Unroll over a sequence (x_0, x_1, x_2):



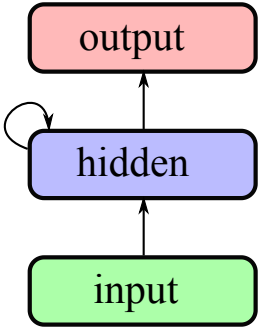
Recurrent Neural Network



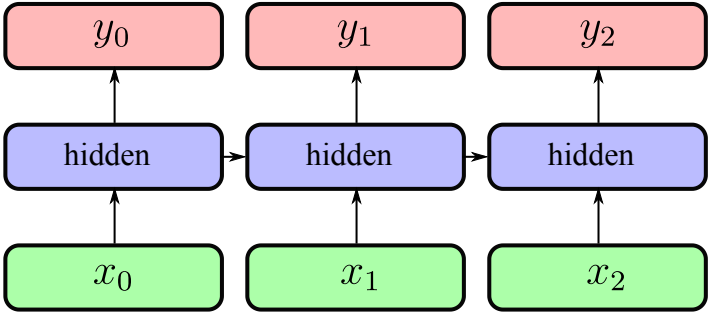
Unroll over a sequence (x_0, x_1, x_2):



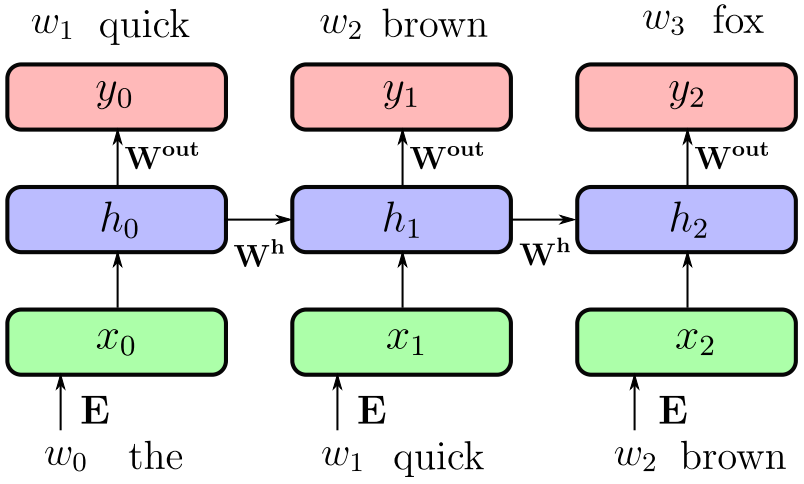
Recurrent Neural Network



Unroll over a sequence (x_0, x_1, x_2):

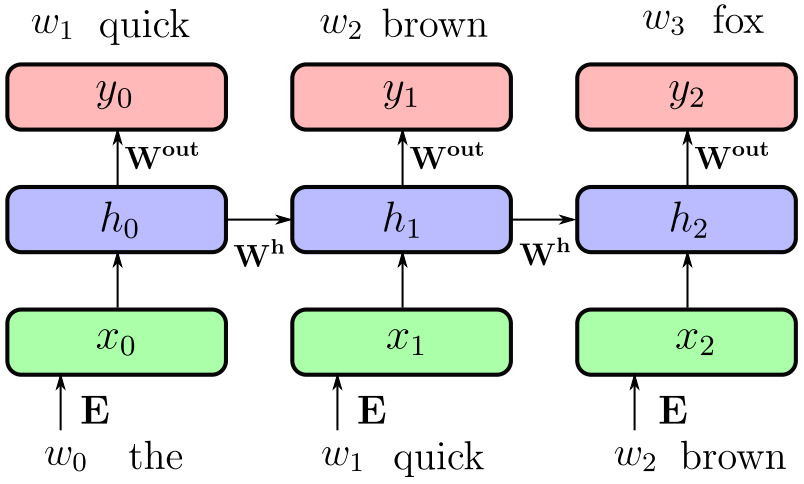


Language Modelling



- input: $(w_0, w_1, ..., w_t)$ = sequence of embeddings words
- output: $(w_1, w_2, ..., w_{t+1})$ = shifted sequence of embedings

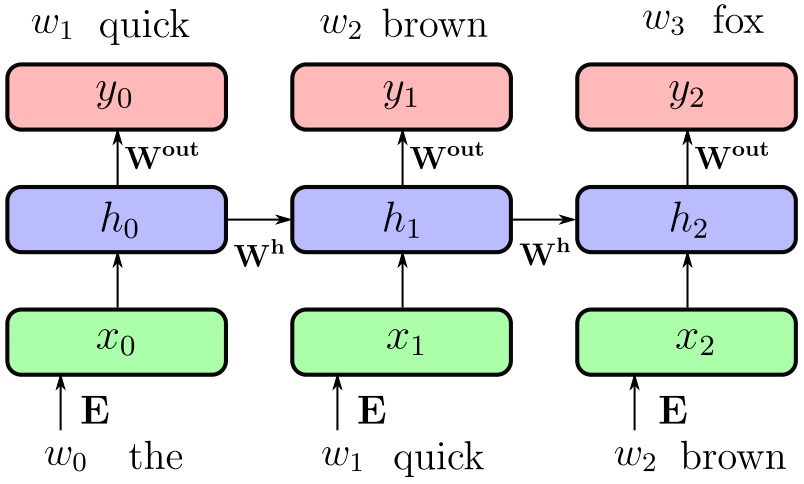
Language Modelling



- $x_t = \text{Embedding}(w_t) \in \mathbb{R}^E$
- $h_t = g(W^h h_{t-1} + W^x x_t + b^h) \in \mathbb{R}^H$ (often $g = \tanh$)
- $y = \text{softmax}(W^o h_t + b^o) \in \Delta_K$ ($K = |V|$ here)

How many parameters ?

Language Modelling



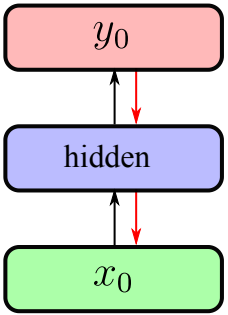
- $x_t = \text{Embedding}(w_t) \in \mathbb{R}^E$
- $h_t = g(W^h h_{t-1} + W^x x_t + b^h) \in \mathbb{R}^H$ (often $g = \tanh$)
- $y = \text{softmax}(W^o h_t + b^o) \in \Delta_K$ ($K = |V|$ here)

How many parameters ?

$$E \times |V| \quad + \quad H \times H + H \times E + H \quad + \quad K \times H + K$$

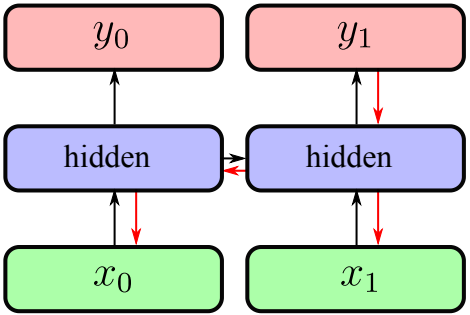
Backpropagation through time

Similar as standard backpropagation on unrolled network



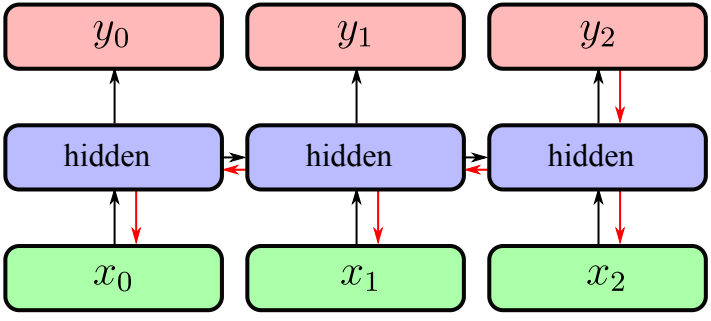
Backpropagation through time

Similar as standard backpropagation on unrolled network



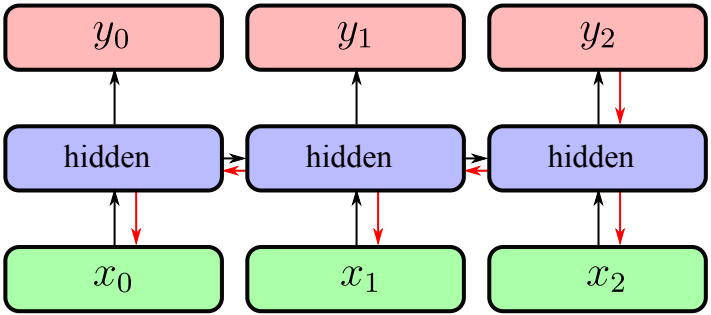
Backpropagation through time

Similar as standard backpropagation on unrolled network



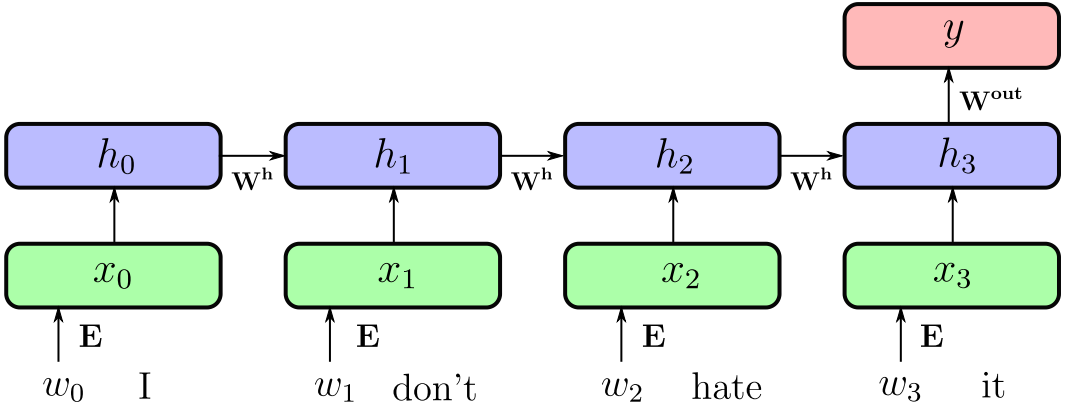
Backpropagation through time

Similar as standard backpropagation on unrolled network



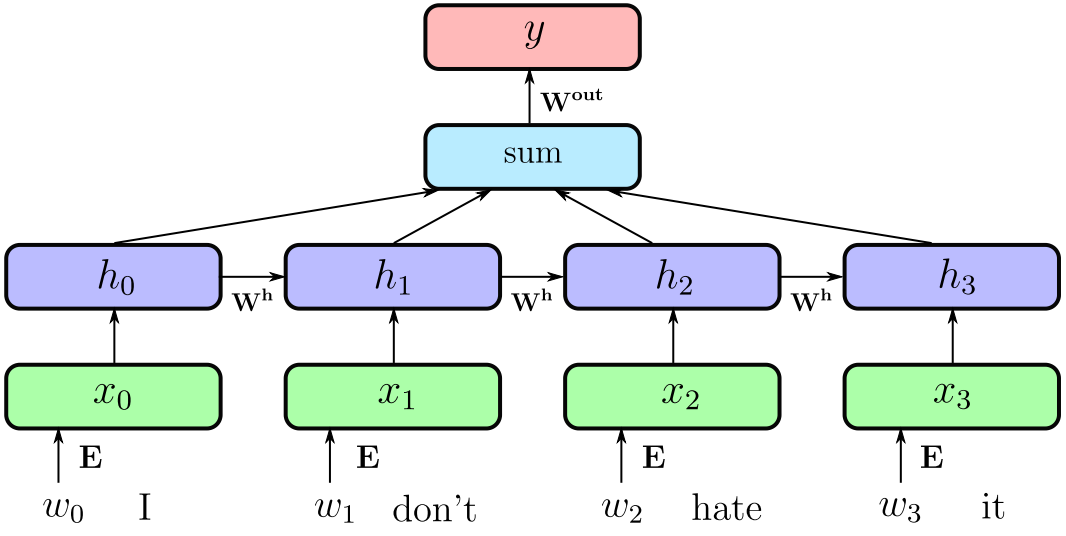
- Similar as training very deep networks with tied parameters
- Example between x_0 and y_2: W^h is used twice
- Usually truncate the backprop after T timesteps
- Difficulties to train long-term dependencies

Other uses: opinion analysis



- Output is opinion (1 for positive, 0 for negative)
- Very dependent on words order
- Very flexible network architectures

Other uses: opinion analysis

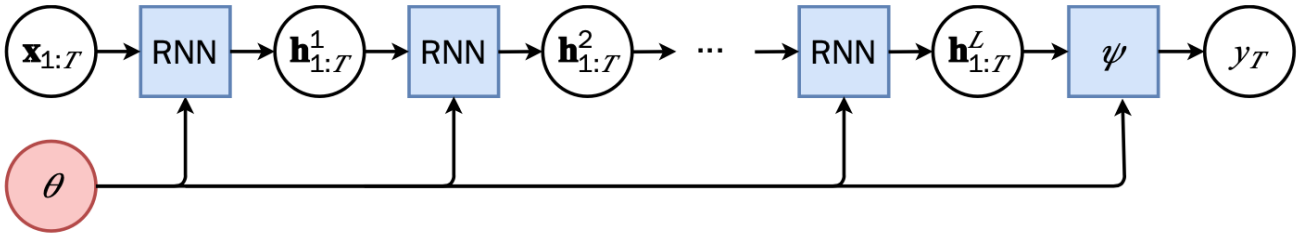


- Output is opinion (1 for positive, 0 for negative)
- Very dependent on words order
- Very flexible network architectures

Stacked RNNs

Recurrent networks can be viewed as layers producing sequences $\mathbf{h}^l_{1:T}$ of activations.

As for dense layers, recurrent layers can be composed in series to form a **stack** of recurrent networks.



Bidirectional RNNs

Computing the recurrent states forward in time does not make use of future input values $\mathbf{x}_{t+1:T}$, even though there are known

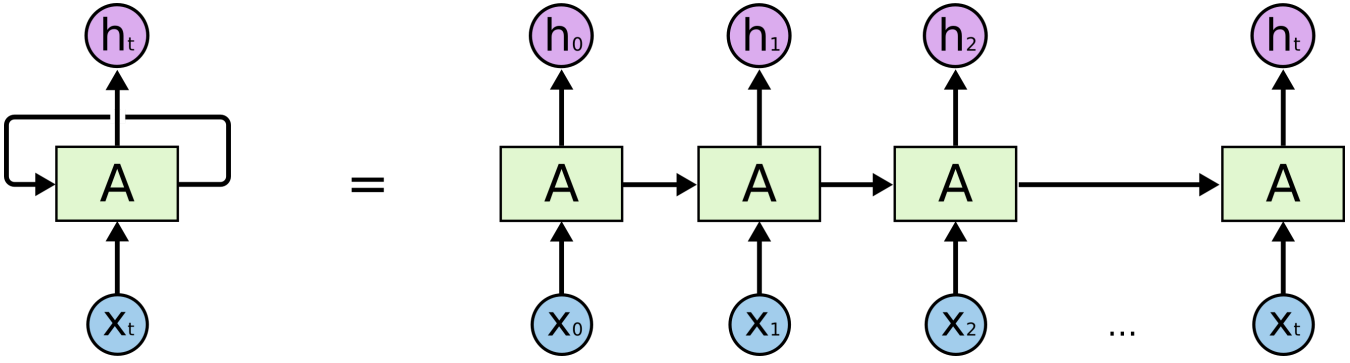
RNNs can be made bidirectional by consuming the sequence in both directions.

Effectively, this amounts to run the same (single direction) RNN twice:

- once over the original sequence $\mathbf{x}_{1:T}$
- once over the reversed sequence $\mathbf{x}_{T:1}$

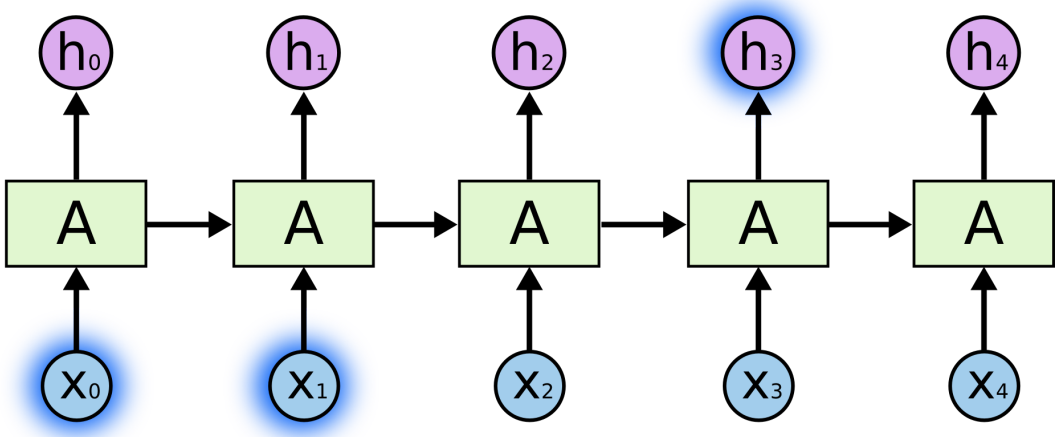
The resulting recurrent states of the bidirectional RNN is the concatenation of two resulting sequences of recurrent states

Long-term dependencies



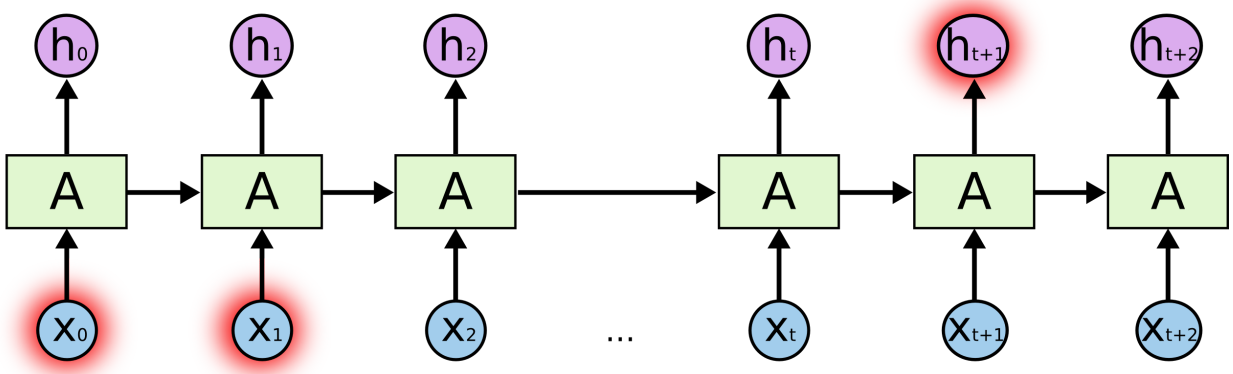
- Nice idea: allows information along a sequence to persist
- Gradient computed through backpropagation through time: backpropagation applied to the RNN unrolled along the sequence
- If the sequence is long: many multiplication with the same matrices!
- Leads to the so-called vanishing or exploding gradient problem

Long-term dependencies



- Small number of matrix products
- Gradients do not vanish or explode
- The network can learn short-term dependencies

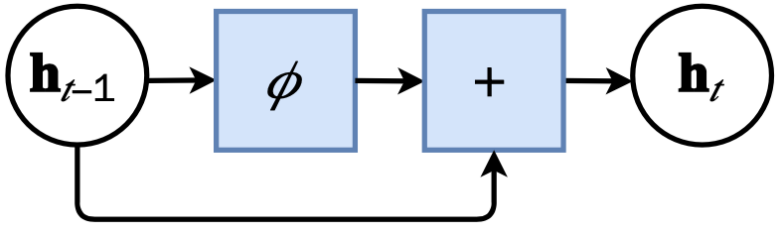
Long-term dependencies



- Longer dependencies require many matrix products: vanishing or exploding gradients makes it hard with a simple RNN
- Introduces a strong bias: the network learns only short-term dependencies
- Tricks: gradient clipping, non-saturating activations, ...
- but mainly more complicated layers using architectures based on gates
- The most popular one is the LSTM

Gating

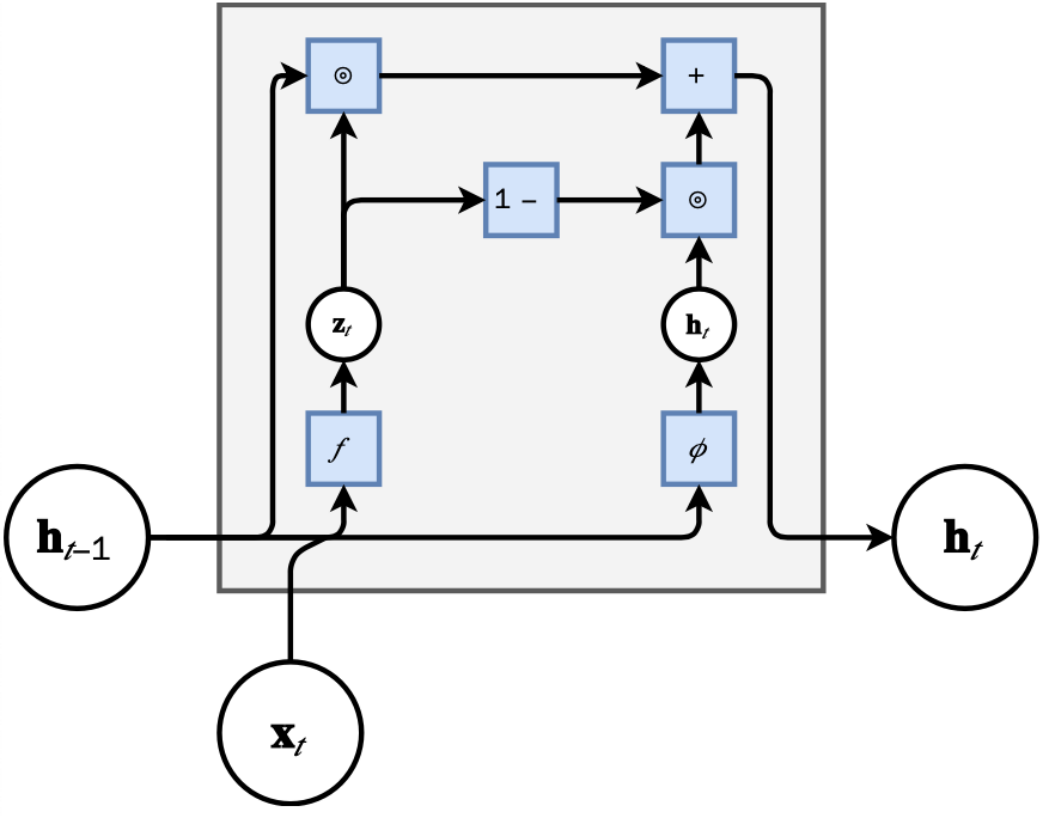
RNN cells can include a pass-through, or additive paths, so that the recurrent state does not go repeatedly through a squashing non-linearity. This is identical to skip connections in ResNet.



For instance, the recurrent state update can be a per-component weighted average of its previous value $\mathbf{h}_{t-1}$ and a full update $\bar{\mathbf{h}}_t$, with the weighting $\mathbf{z}_t$ depending on the input and the recurrent state, hence acting as a forget gate.

$$\begin{aligned}
\bar{\mathbf{h}}_t &= \phi(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta) \\
\mathbf{z}_t &= f(\mathbf{x}_t, \mathbf{h}_{t-1}; \theta) \\
\mathbf{h}_t &= \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \bar{\mathbf{h}}_t.
\end{aligned}$$

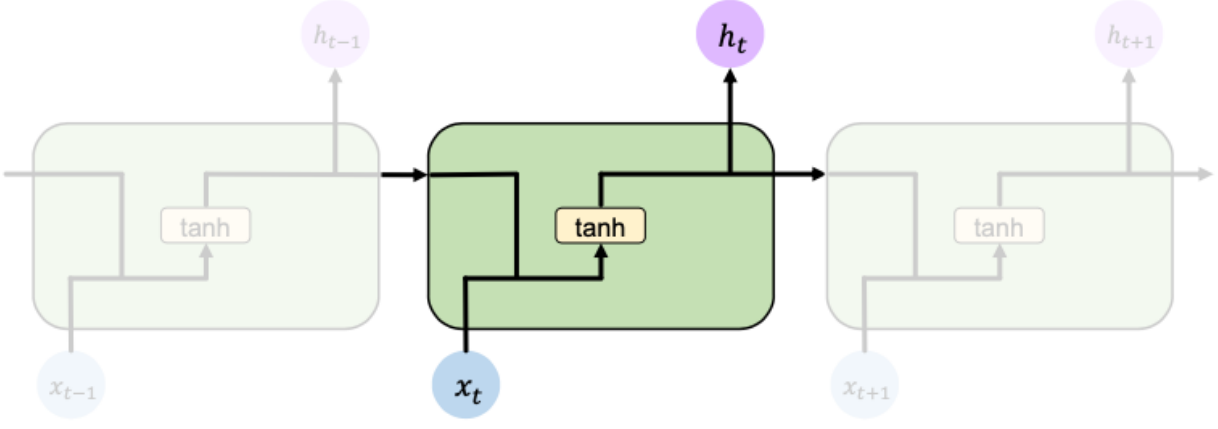
Gating



Long Short Term Memory (LSTM)

Standard RNN

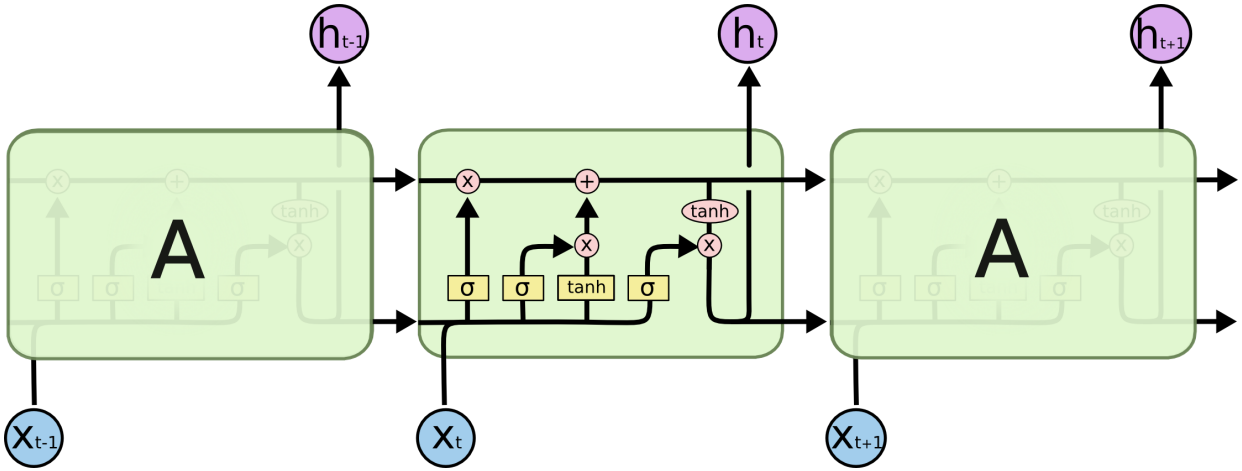
In a standard RNN, the recurrent layer contains a simple computation node



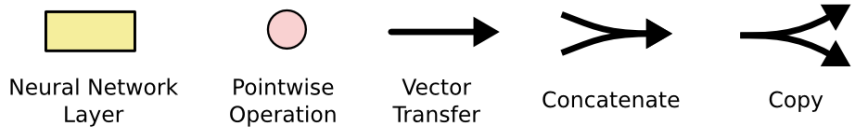
$$h_t = \tanh(W^h h_{t-1} + W^x x_t + b^h) \in \mathbb{R}^H$$

Long Short Term Memory (LSTM)

LSTM contains interacting layers that control information flow via gates



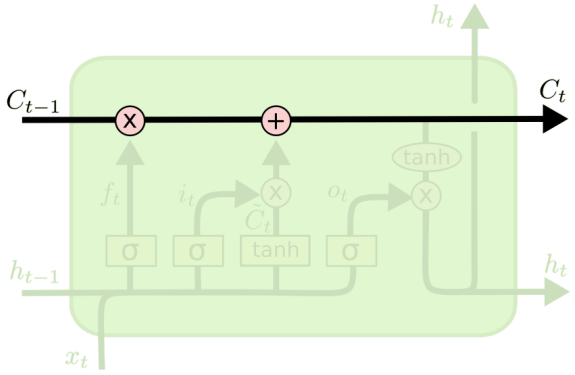
LSTM layers are able to track or memorize information throughout many time steps



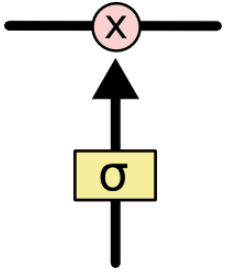
Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 1997

Long Short Term Memory (LSTM)

A LSTM layer uses a cell state c_t where it's easy for information to flow



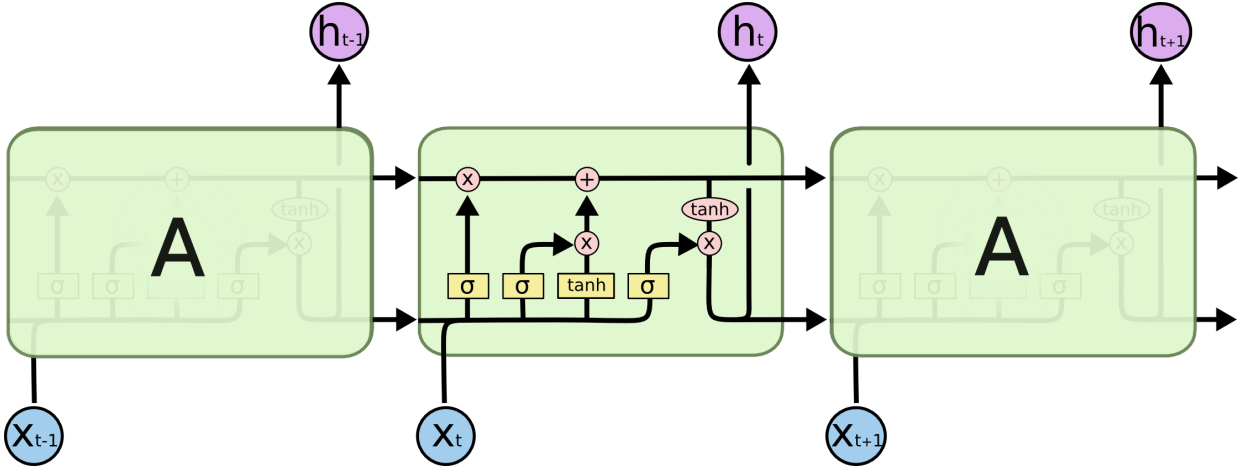
Information is added or removed to cell state through structures called gates



Gates optionally let information through, via a sigmoid activation and pointwise product

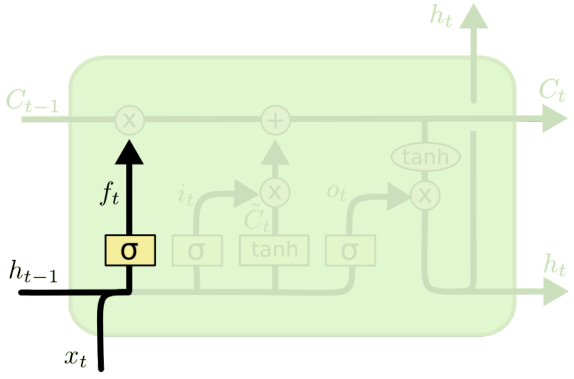
Long Short Term Memory (LSTM)

How do LSTM work ?



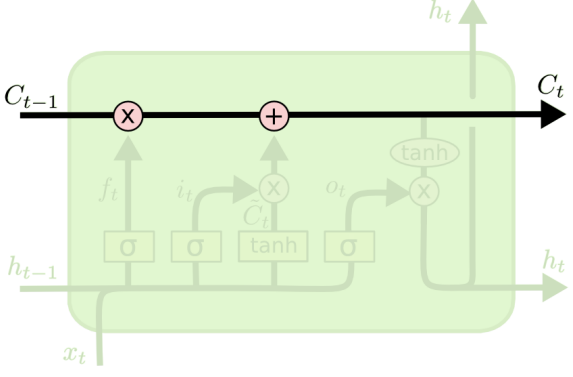
Long Short Term Memory (LSTM)

LSTM learns to forget irrelevant parts of the previous state



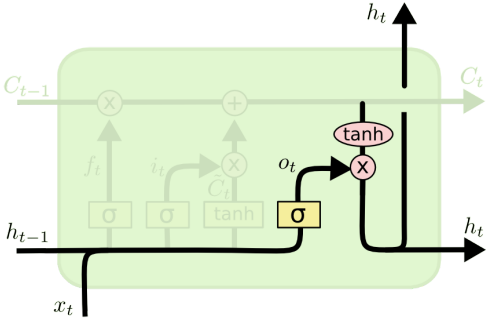
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

LSTM learns to selectively update cell state c_t values



Long Short Term Memory (LSTM)

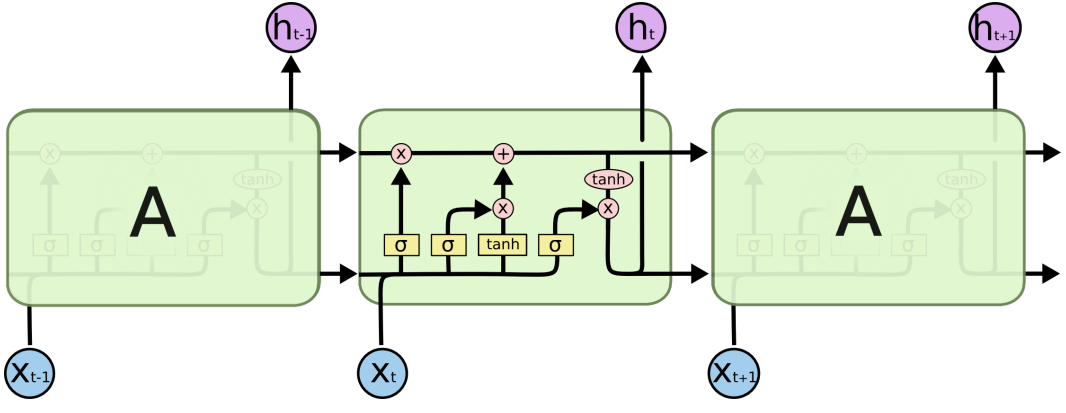
LSTM learns an output gate that outputs certain parts of the cell state



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

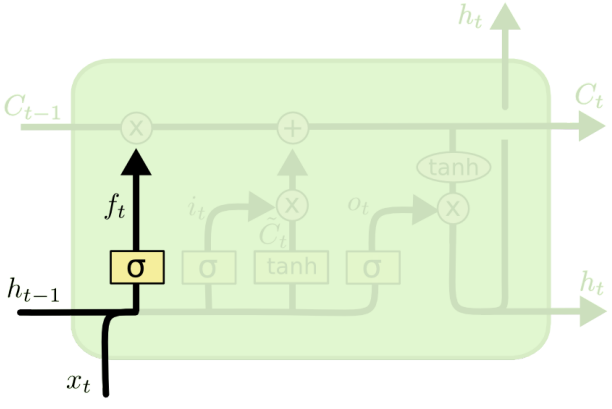
$$h_t = o_t * \tanh\left(C_t\right)$$

A LSTM learns to forget, update and output



LSTM

Forget **irrelevant information**

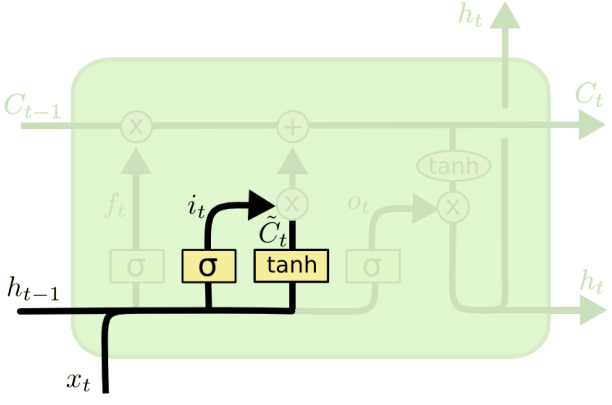


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

- Uses previous hidden state h_{t-1} and input x_t
- Learns to completely forget or keep previous cell state c_{t-1} through a layer with sigmoid σ activation $\in [0, 1]$

LSTM

Identify new information to be stored



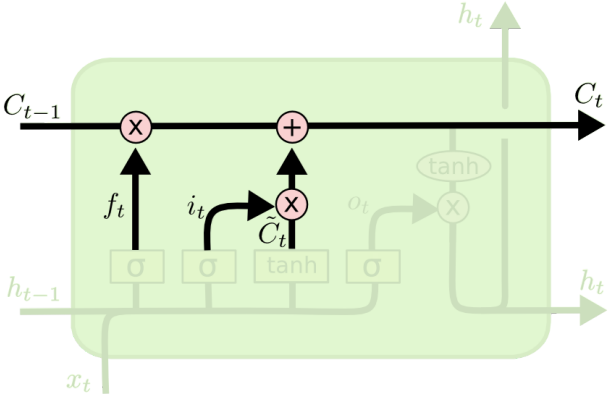
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

- Uses previous hidden state h_{t-1} and input x_t
- σ layer: decide what values to update
- $\tanh$ layer: generate new vector of candidate values that could be added to the state

LSTM

Update **cell state**



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

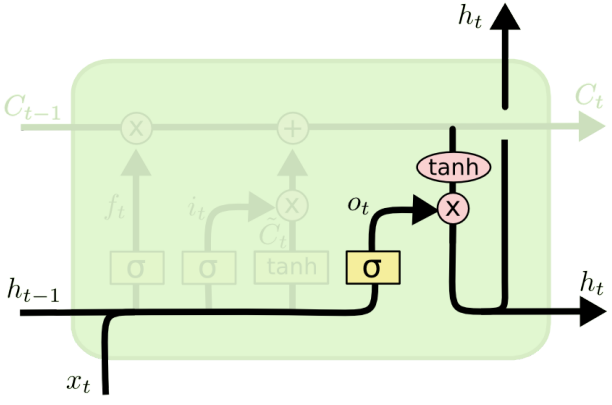
Apply forget operation to the previous cell state C_{t-1} using $f_t * C_{t-1}$

Add new candidate values, scaled by how much we decided to update : $i_t * \tilde{C}_t$

Combine the two to produce new cell state C_t

LSTM

output **hidden state**



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

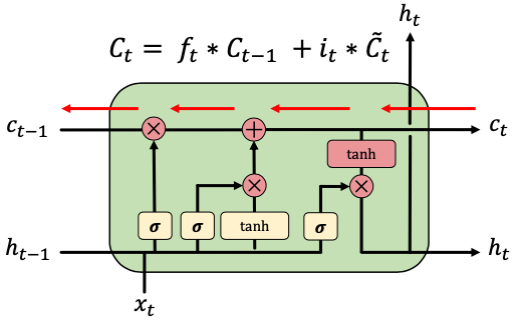
Construct hidden state h_t (output of the recurrent layer) as a filtered version of the cell state c_t

σ layer: decide what parts of state c_t to output

$\tanh$ layer : squash values of cell state in $[-1, 1]$

$o_t * \tanh(C_t)$: output filtered version of cell state

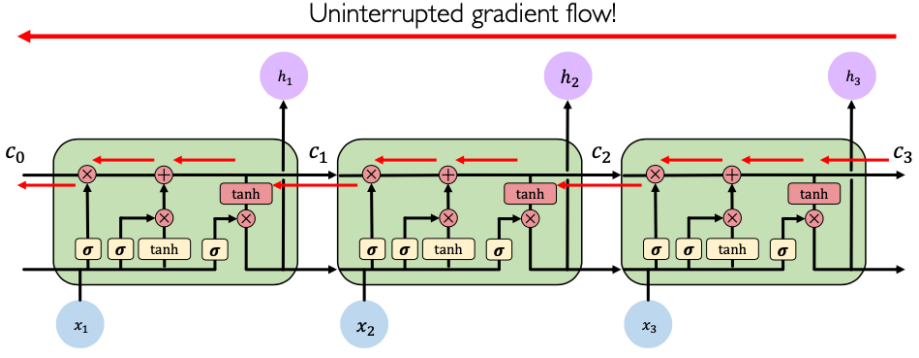
LSTM gradient flow



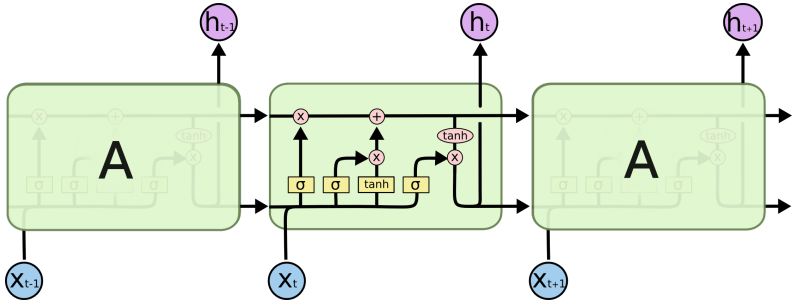
Backpropagation from C_t to C_{t-1} requires only element-wise multiplication !

No matrix multiplication: avoids the vanishing gradient problem

Leads to an uninterrupted gradient flow!

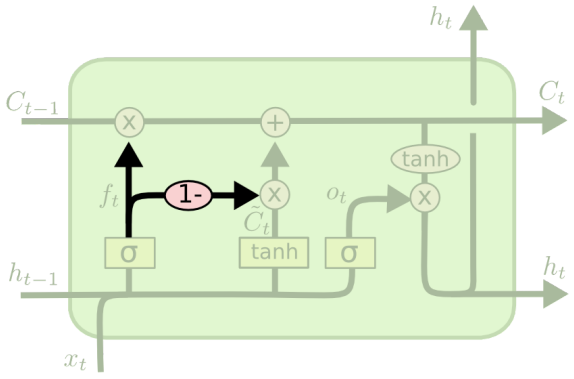


LSTM key ingredients



1. Maintain a separate cell state c_t in addition to a hidden state h_t
2. Use gates to control the flow of information
   - Forget gate to learn to keep information from previous cell state C_{t-1}
   - Input gate to learn to store new information in the new cell state C_t
   - Output gate to learn to return the hidden state
3. Backpropagation from C_t to C_{t-1} does not require matrix multiplication: no vanishing gradient problem

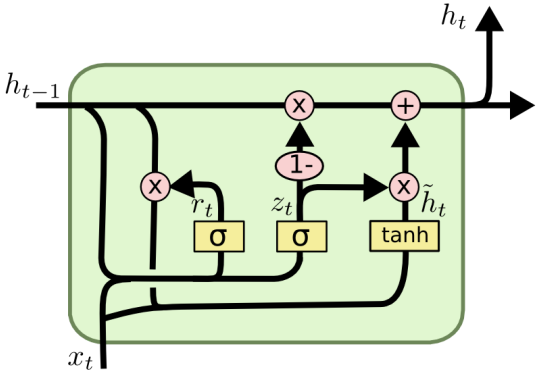
Around LSTM: tied forget and input gates



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

- Coupled or tied forget and input gates
- Decide what should be kept and forgotten at the same time

Around LSTM: GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- A very popular alternative to the LSTM
- Combines forget and input gates into a single update gate
- Also, it merges the cell and hidden states
- Simpler than LSTM and often performs the same

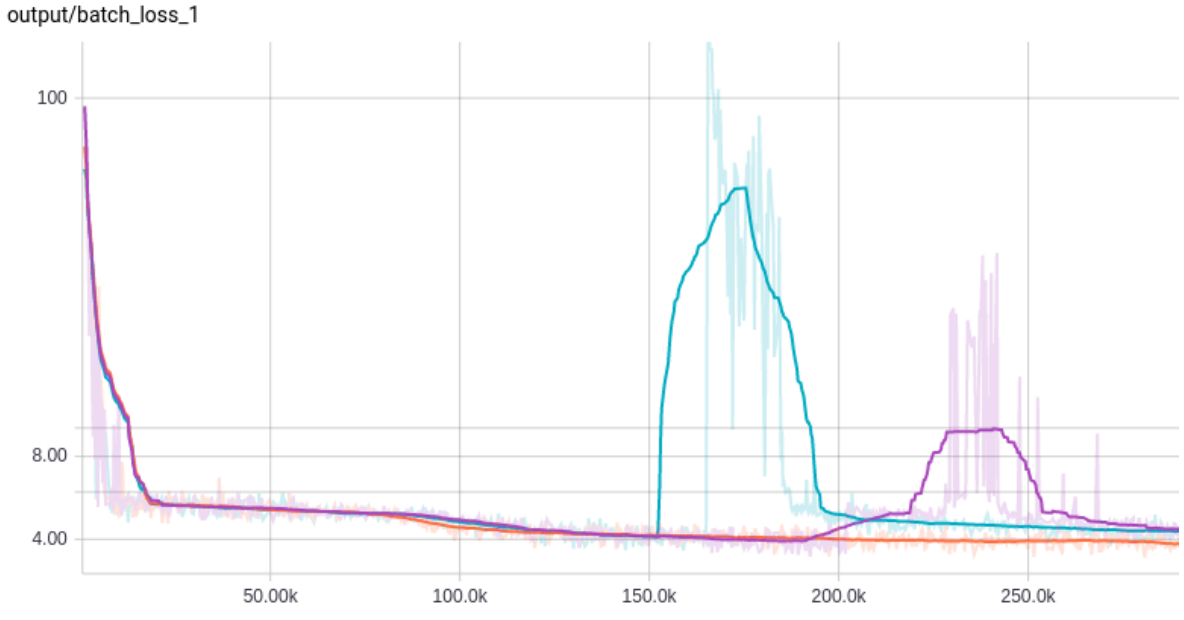
Many other variants!

But main recurrent layers in `PyTorch` are `rnn`, `lstm` and `gru` layers

Exploding gradients

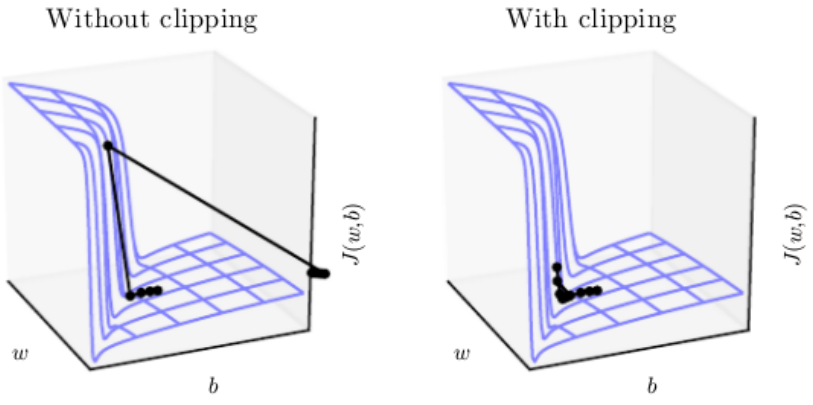
Exploding gradients

Gated units prevent gradients from vanishing, but not from exploding.



Credits: pat-coady.

Gradient clipping



The standard strategy to solve this issue is gradient norm clipping, which rescales the norm of the gradient to a fixed threshold δ when it is above:

$$\tilde{\nabla} f = \frac{\nabla f}{||\nabla f||} \min(||\nabla f||, \delta).$$

Orthogonal initialization

Let us consider a simplified RNN, with no inputs, no bias, an identity activation function σ (as in the positive part of a ReLU) and the initial recurrent state $\mathbf{h}_0$ set to the identity matrix.

We have,

$$\begin{aligned}\mathbf{h}_t &= \sigma\ \ \mathbf{W}_{xh}^T\mathbf{x}_t + \mathbf{W}_{hh}^T\mathbf{h}_{t-1} + \mathbf{b}_h \\ &= \mathbf{W}_{hh}^T\mathbf{h}_{t-1} \\ &= \mathbf{W}^T\mathbf{h}_{t-1}.\end{aligned}$$

For a sequence of size n, it comes

$$\mathbf{h}_n = \mathbf{W}(\mathbf{W}(\mathbf{W}(...(\mathbf{W}\mathbf{h}_0)...))) = \mathbf{W}^n\mathbf{h}_0 = \mathbf{W}^n I = \mathbf{W}^n.$$

Ideally, we would like $\mathbf{W}^n$ to neither vanish nor explode as n increases.

Orthogonal initialization

Theorem

Let $\rho(\mathbf{A})$ be the spectral radius of the matrix $\mathbf{A}$, defined as

$$\rho(\mathbf{A}) = \max\{|\lambda_1|, ..., |\lambda_d|\}.$$

We have:

- if $\rho(\mathbf{A}) < 1$ then $\lim_{n\to\infty} ||\mathbf{A}^n|| = \mathbf{0}$ (= vanishing activations),
- if $\rho(\mathbf{A}) > 1$ then $\lim_{n\to\infty} ||\mathbf{A}^n|| = \infty$ (= exploding activations).

Orthogonal initialization

If $\mathbf{A}$ is orthogonal, then it is diagonalizable and all its eigenvalues are equal to -1 or 1. In this case, the norm of $\mathbf{A}^n = \mathbf{S}\Lambda^n\mathbf{S}^{-1}$ remains bounded.

- Therefore, initializing $\mathbf{W}$ as a random orthogonal matrix will guarantee that activations will neither vanish nor explode.
- In practice, a random orthogonal matrix can be found through the SVD decomposition or the QR factorization of a random matrix.
- This initialization strategy is known as orthogonal initialization.

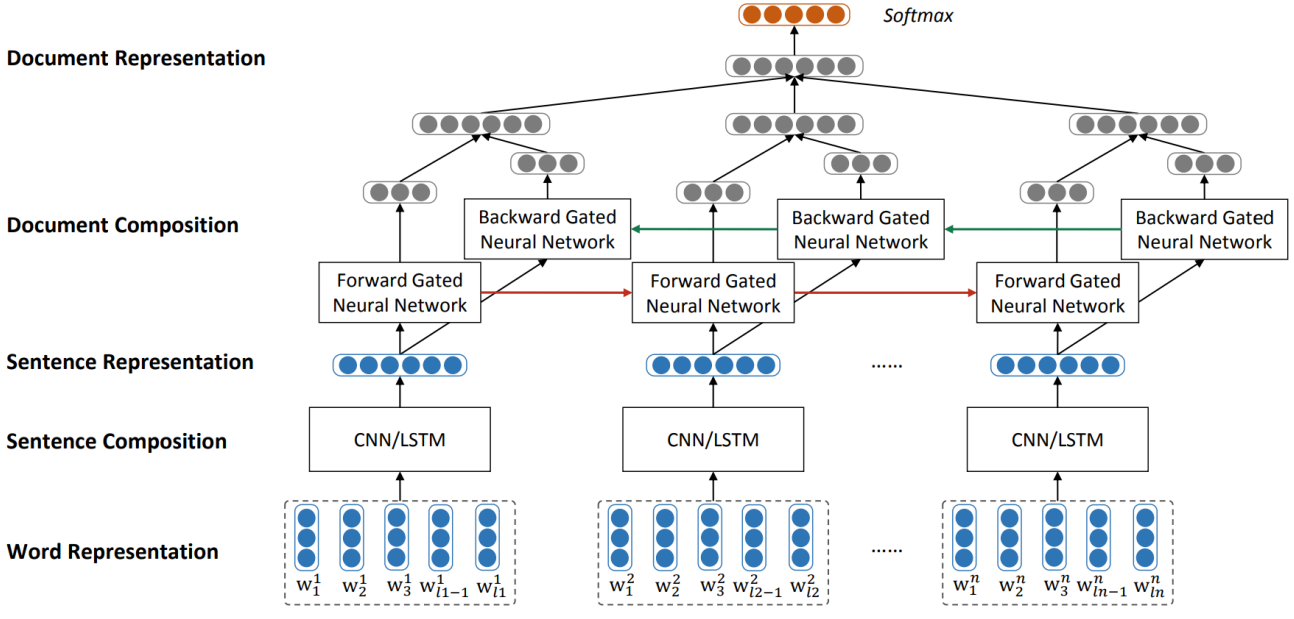
Exploding activations are also the reason why squashing non-linearity functions (such as $\tanh$) are preferred in RNNs.

- They avoid recurrent states from exploding by upper bounding $||\mathbf{h}_t||$.
- (At least when running the network forward.)

Applications

(Only some)

Sentiment analysis



Document-level modeling for sentiment analysis (= text classification), with stacked, bidirectional and gated recurrent networks.

Credits: Duyu Tang et al, Document Modeling with Gated Recurrent Neural Network for Sentiment Classification, 2015.

Language models

Model language as a Markov chain, such that sentences are sequences of words $\mathbf{w}_{1:T}$ drawn repeatedly from

$$p(\mathbf{w}_t | \mathbf{w}_{1:t-1}).$$

This is an instance of sequence synthesis, for which predictions are computed at all time steps t.

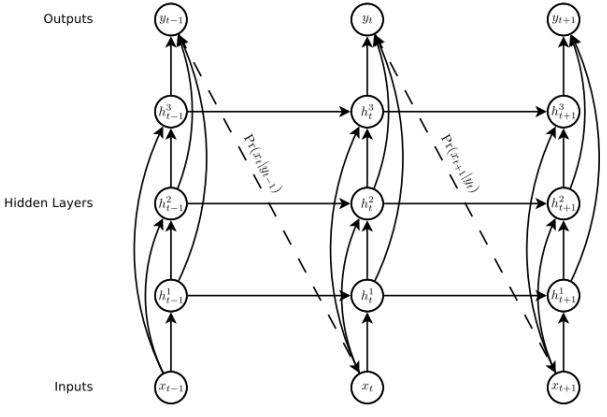


Figure 1: **Deep recurrent neural network prediction architecture.** The circles represent network layers, the solid lines represent weighted connections and the dashed lines represent predictions.

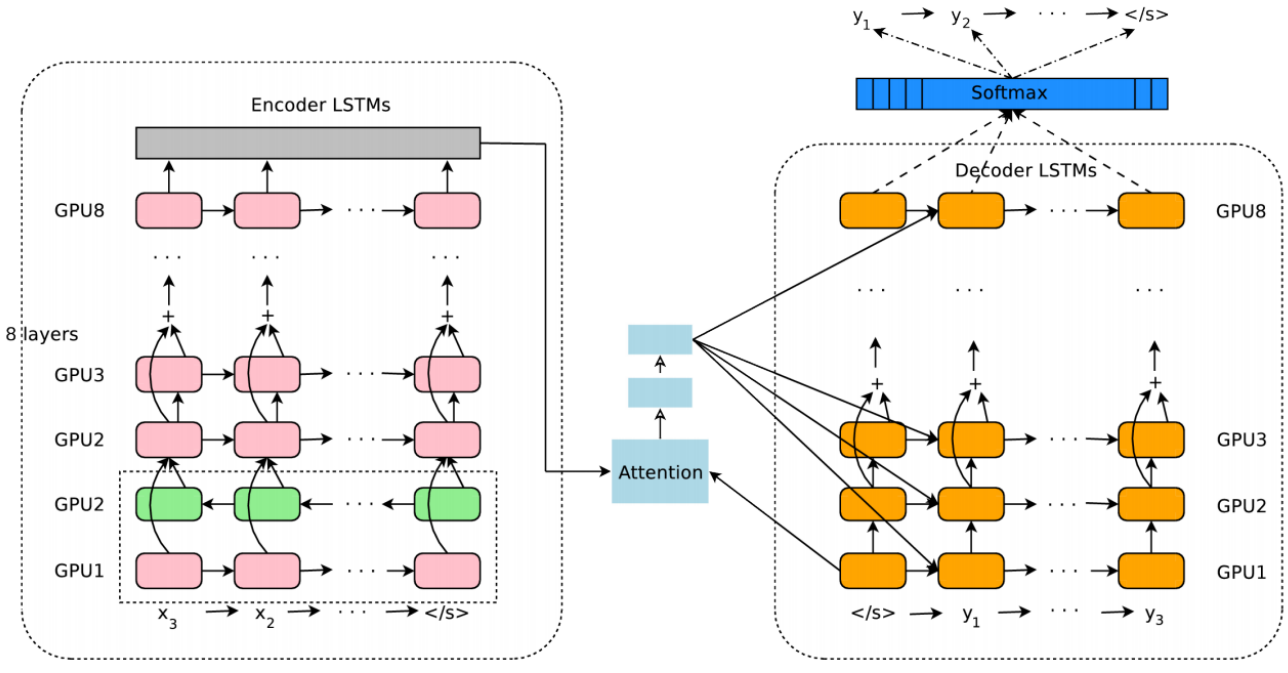
Credits: Alex Graves, Generating Sequences With Recurrent Neural Networks, 2013.

Sequence synthesis

The same generative architecture applies to any kind of sequences.

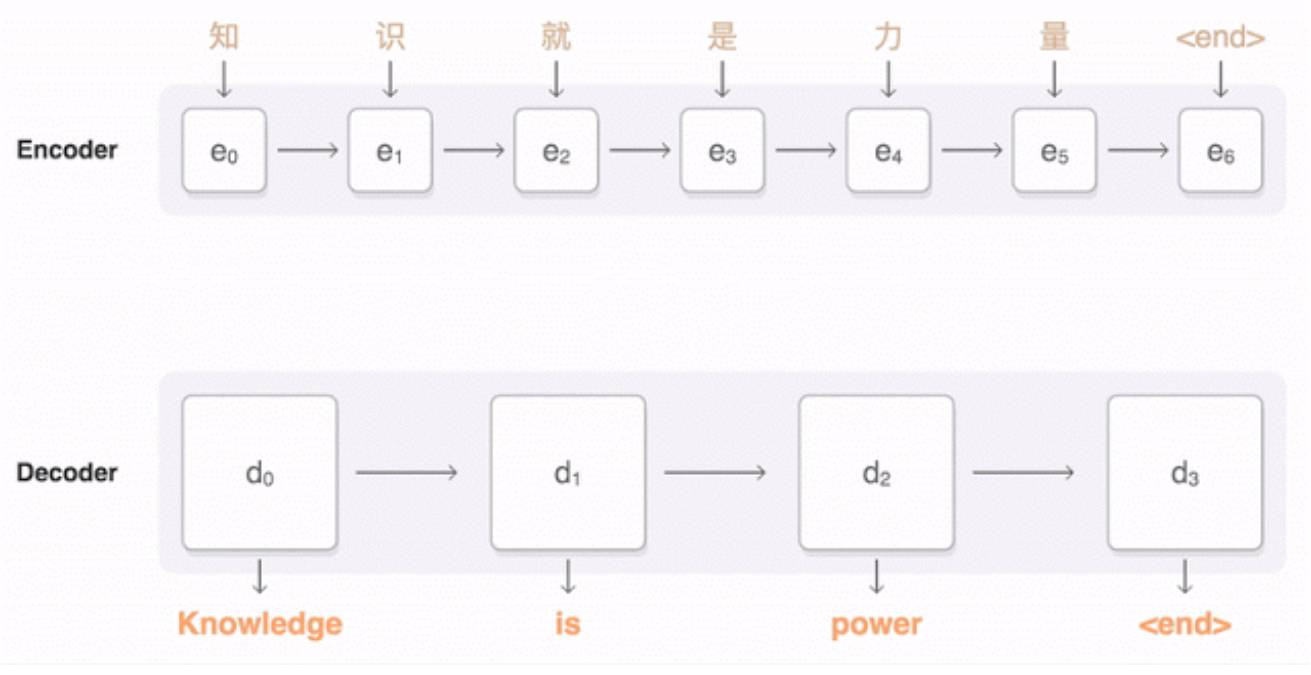
E.g., `sketch-rnn-demo` for sketches defined as sequences of strokes.

Neural machine translation



Credits: Yonghui Wu et al, Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016.

Neural machine translation



Credits: Yonghui Wu et al, Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016.

Text-to-speech synthesis

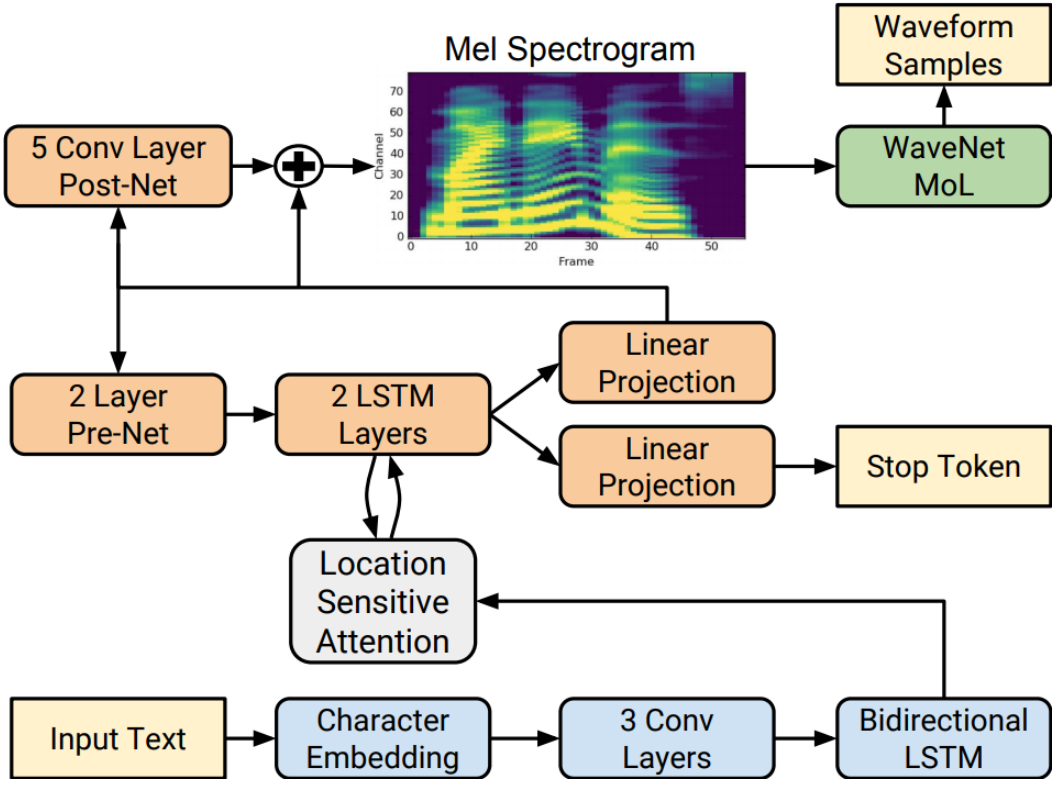
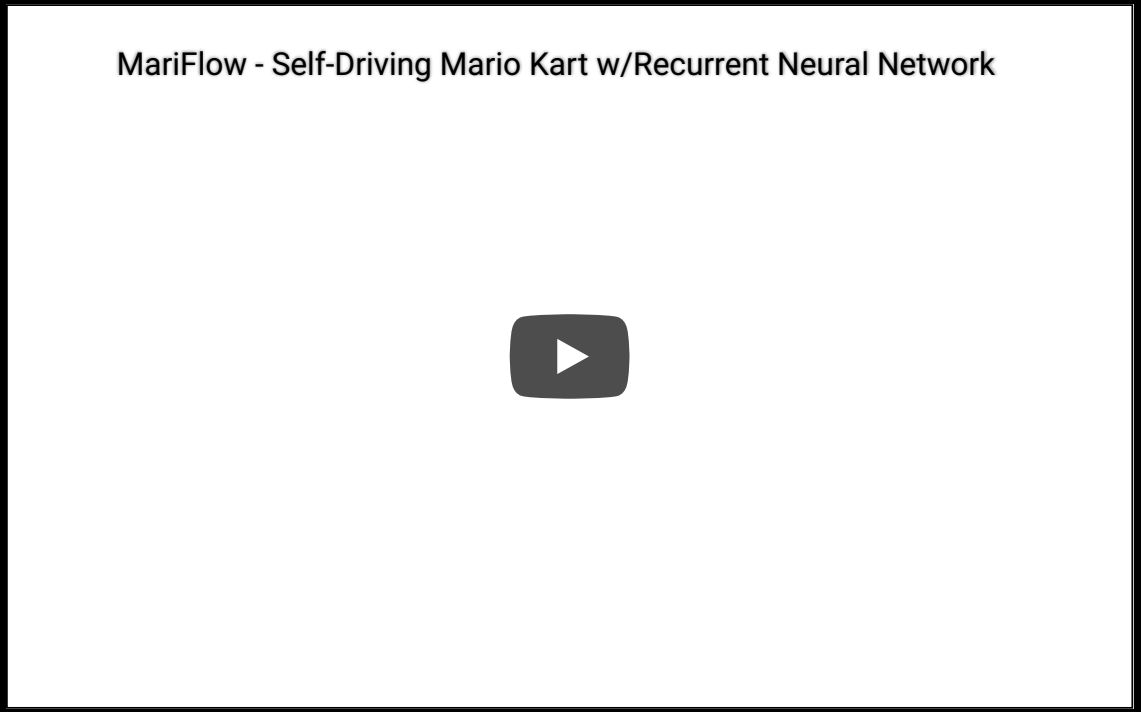


Image credits: Shen et al, 2017. arXiv:1712.05884.



Learning to control

A recurrent network playing Mario Kart

Beyond sequences

*An increasingly large number of **people are defining the networks procedurally in a data-dependent way (with loops and conditionals)**, allowing them to change dynamically as a function of the input data fed to them. It's really **very much like a regular program, except it's parameterized**.*

Yann LeCun (Director of AI Research, Facebook, 2018)

Neural computers

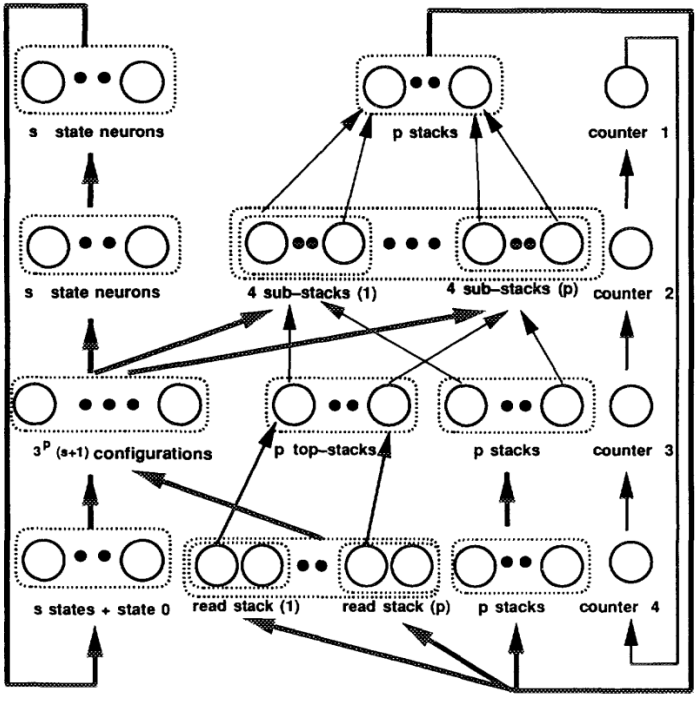
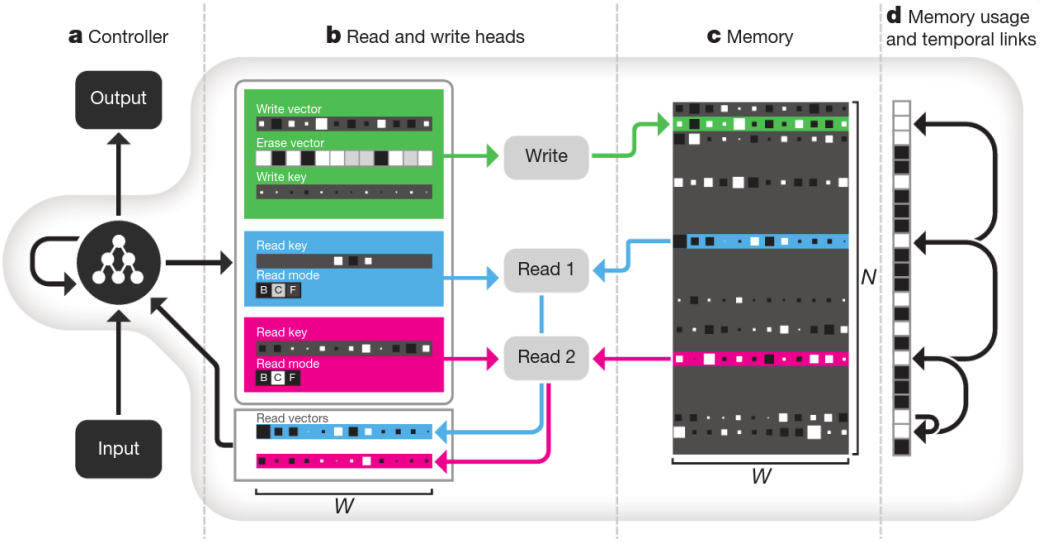


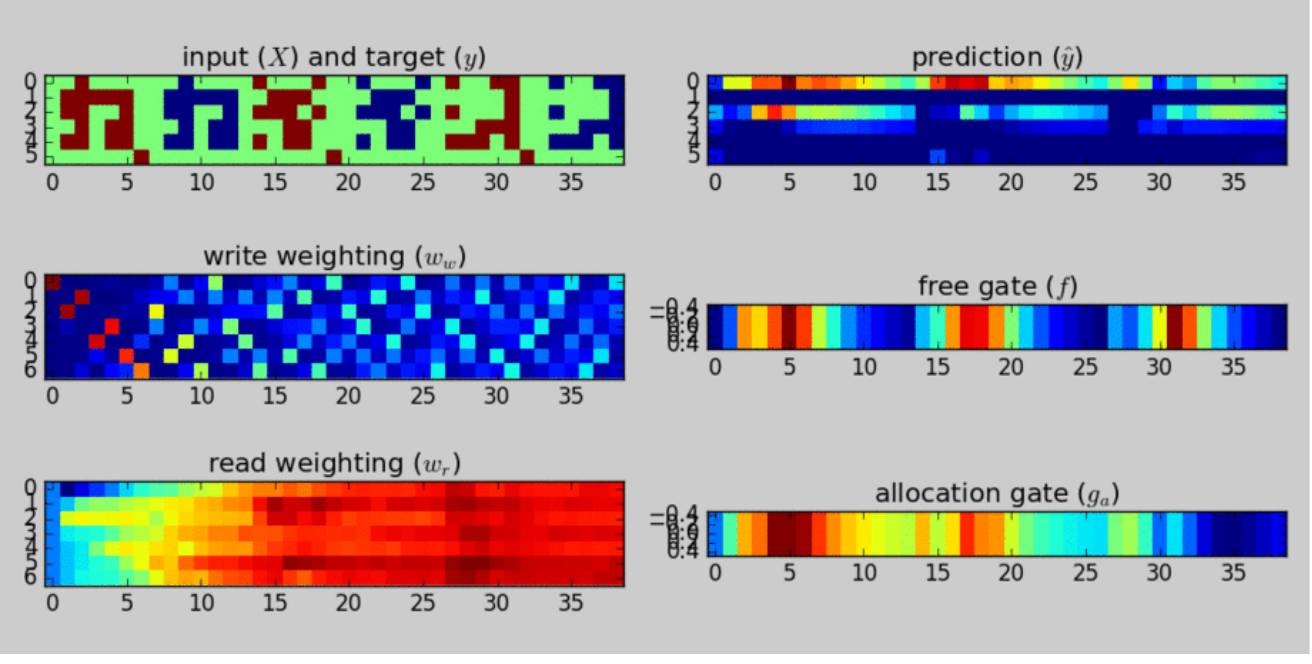
FIG. 1. The universal network.

Any Turing machine can be simulated by a recurrent neural network (Siegelmann and Sontag, 1995)



Networks can be coupled with memory storage to produce neural computers:

- The controller processes the input sequence and interacts with the memory to generate the output.
- The read and write operations attend to all the memory addresses.



A differentiable neural computer being trained to store and recall dense binary numbers. Upper left: the input (red) and target (blue), as 5-bit words and a 1 bit interrupt signal. Upper right: the model's output

Programs as neural nets

The topology of a recurrent network unrolled through time is dynamic.

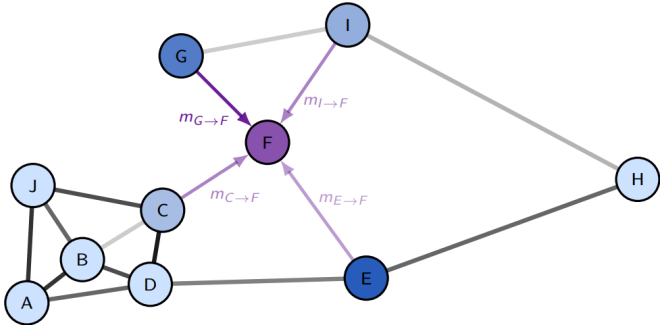
It depends on:

- the input sequence and its size
- a graph construction algorithms which consumes input tokens in sequence to add layers to the graph of computation.

This principle generalizes to:

- arbitrarily structured data (e.g., sequences, trees, graphs)
- arbitrary graph of computation construction algorithm that traverses these structures (e.g., including for-loops or recursive calls).

Neural message passing



Algorithm 1 Message passing neural network

Require: $N \times D$ nodes $\mathbf{x}$, adjacency matrix A

  $\mathbf{h} \leftarrow \text{Embed}(\mathbf{x})$

  for $t = 1, \ldots, T$ **do**

    $\mathbf{m} \leftarrow \text{Message}(A, \mathbf{h})$

    $\mathbf{h} \leftarrow \text{VertexUpdate}(\mathbf{h}, \mathbf{m})$

  end for

  $\mathbf{r} = \text{Readout}(\mathbf{h})$

  return $\text{Classify}(\mathbf{r})$

Even though the graph topology is dynamic, the unrolled computation is fully differentiable. The program is trainable.

Credits: Henrion et al, 2017.

Graph neural network for object detection in point clouds

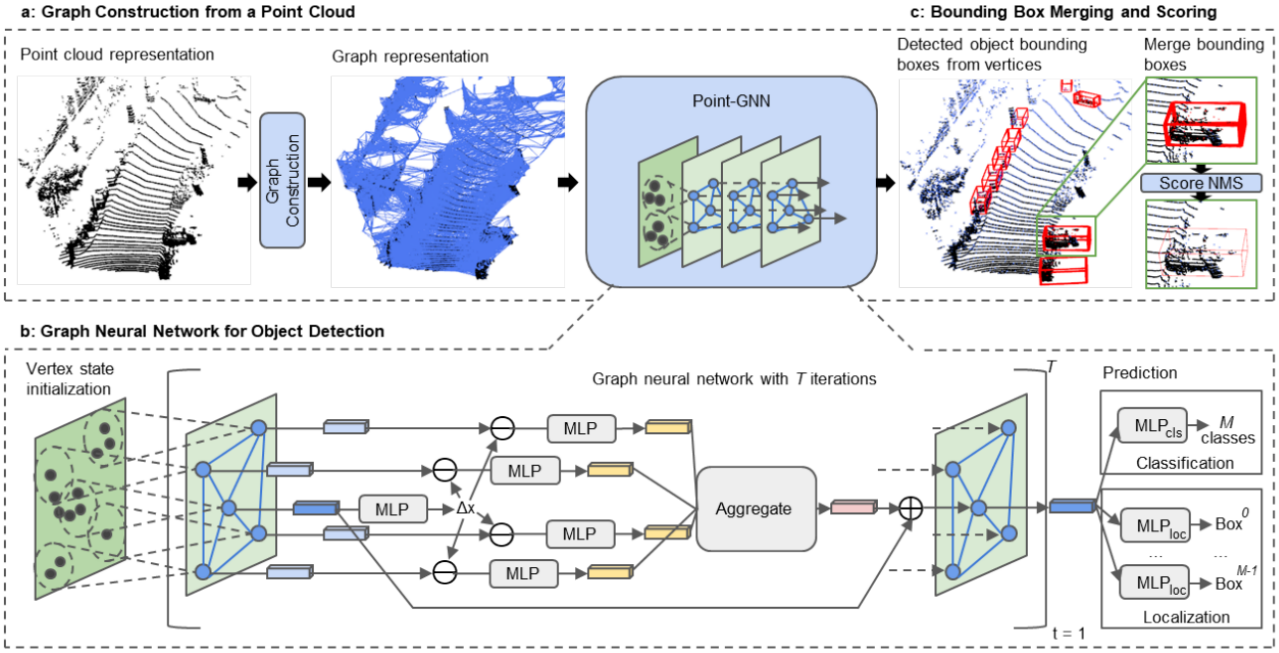
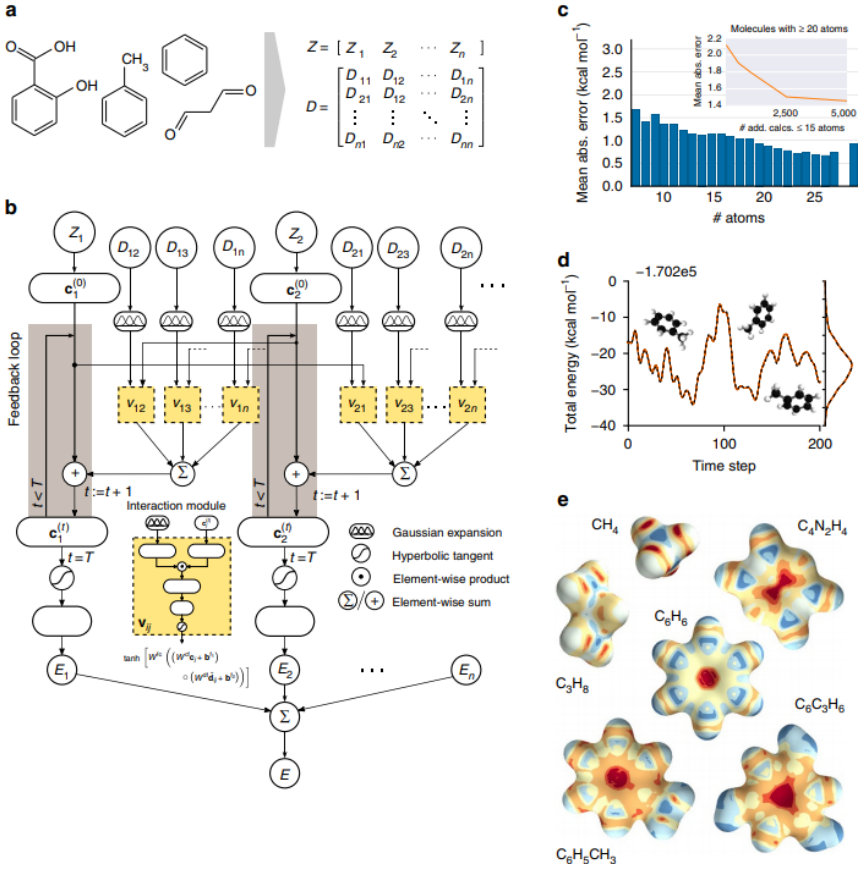


Figure 2. The architecture of the proposed approach. It has three main components: (a) graph construction from a point cloud, (b) a graph neural network for object detection, and (c) bounding box merging and scoring.

Credits: Shi and Rajkumar, Point-GNN, 2020.

Quantum chemistry with graph networks



Credits: Schutt et al, 2017.

Learning to simulate physics with graph networks

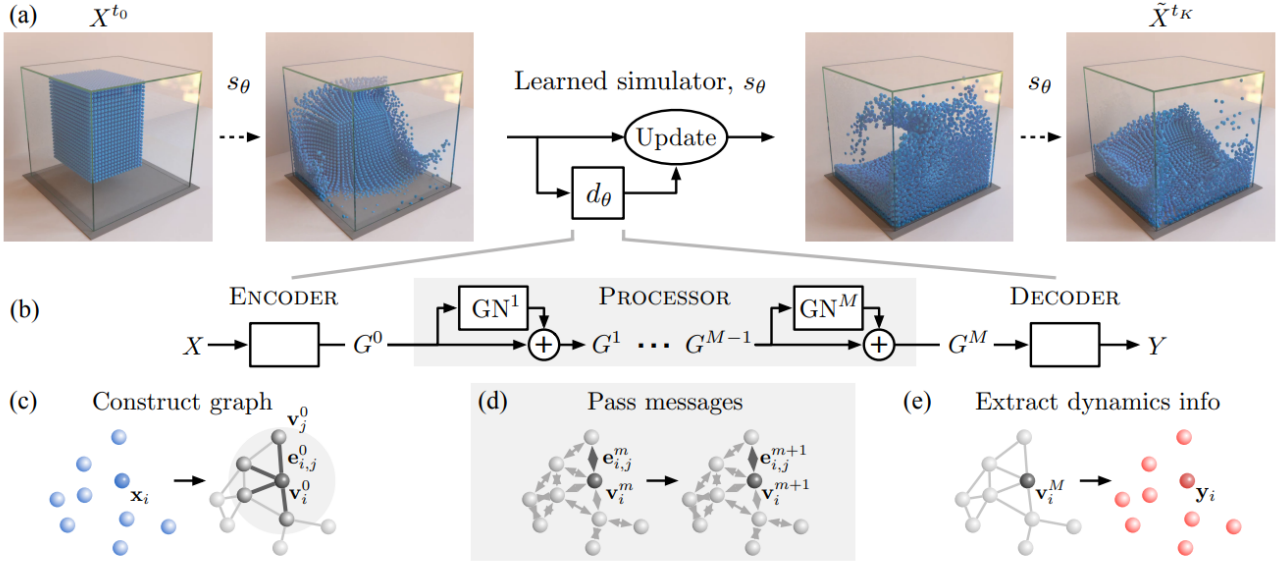
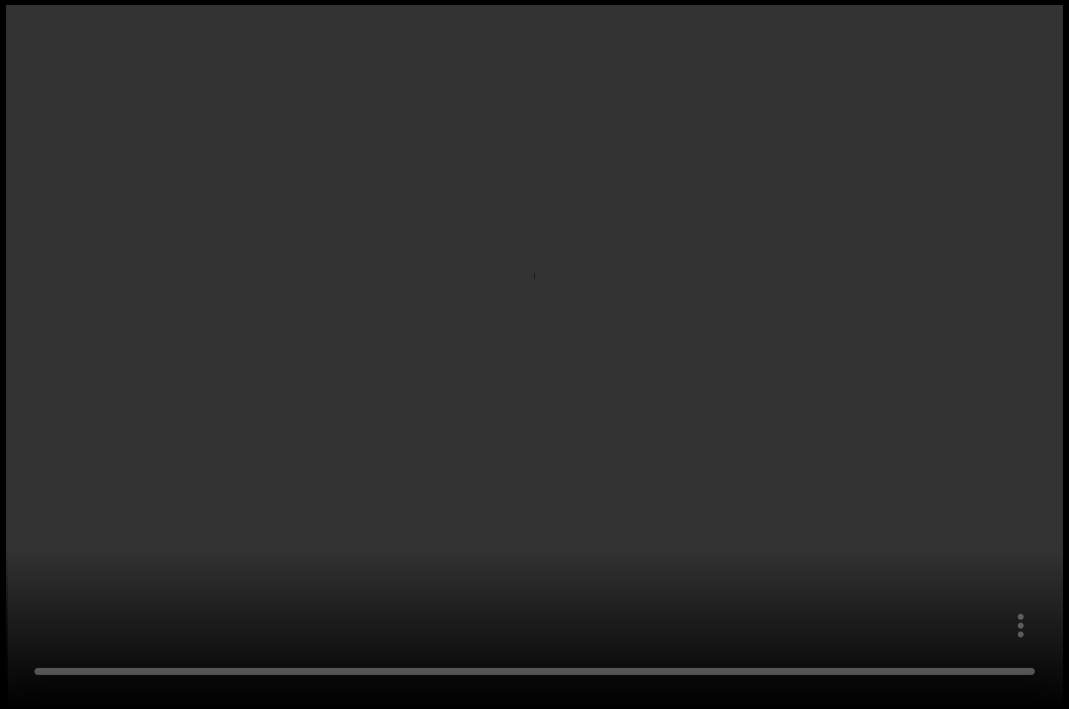


Figure 2. **(a)** Our GNS predicts future states represented as particles using its learned dynamics model, d_θ, and a fixed update procedure. **(b)** The d_θ uses an "encode-process-decode" scheme, which computes dynamics information, Y, from input state, X. **(c)** The ENCODER constructs latent graph, G^0, from the input state, X. **(d)** The PROCESSOR performs M rounds of learned message-passing over the latent graphs, $G^0, \ldots, G^M$. **(e)** The DECODER extracts dynamics information, Y, from the final latent graph, G^M.

Credits: Sanchez-Gonzalez et al, 2020.



Credits: Sanchez-Gonzalez et al, 2020.

Thank you !