

DEEP LEARNING

Lecture 7: Basics of Recurrent Neural Networks

Dr. Yang Lu

Department of Computer Science and Technology

luyang@xmu.edu.cn



RNN Applications

GT: 4 Prediction: 4

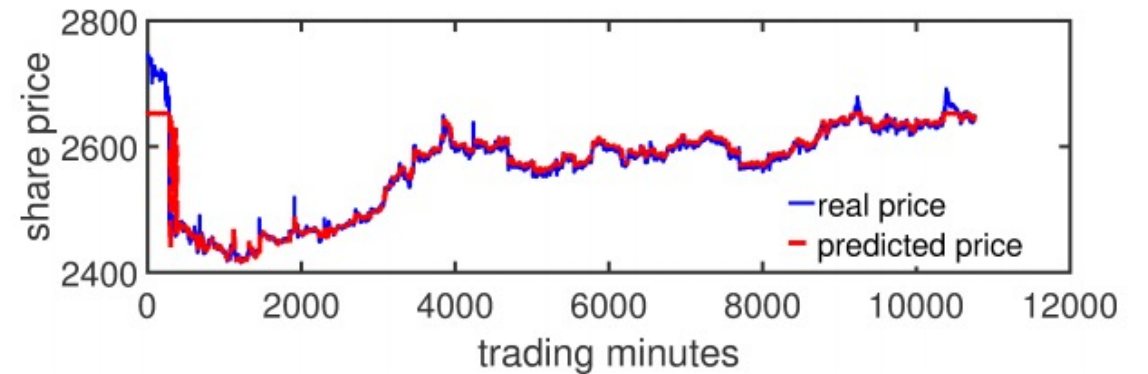
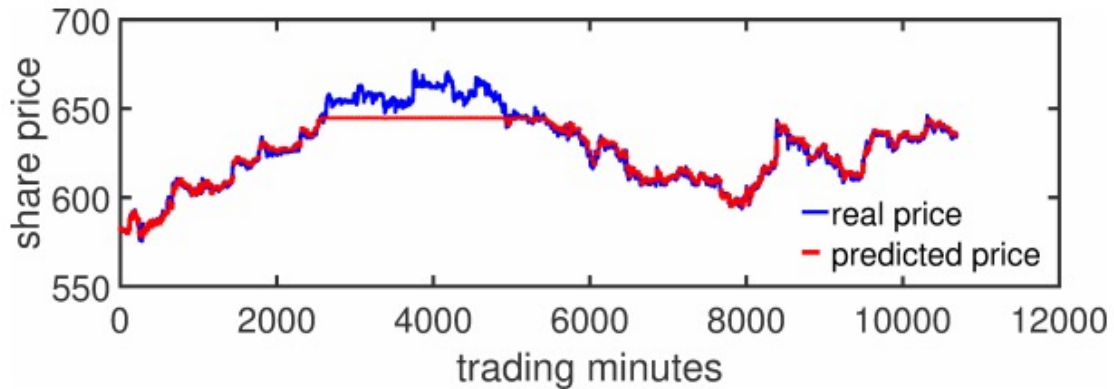
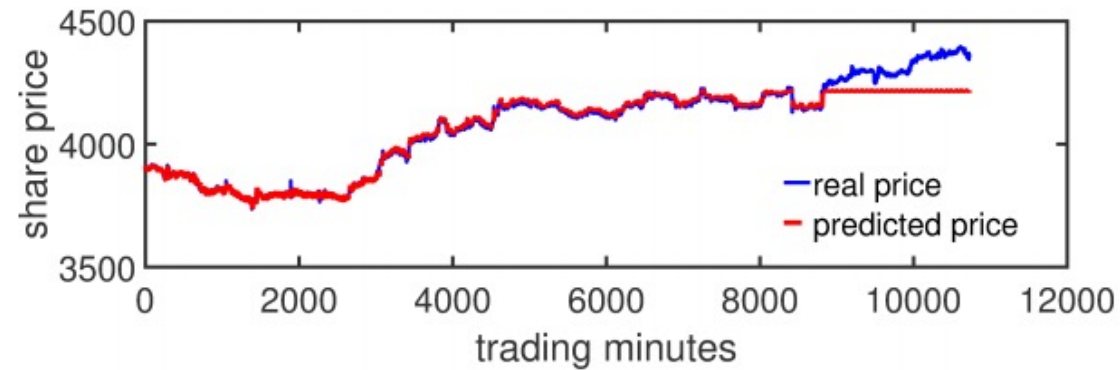
pork belly = delicious .
scallops ?
i do n't .
even .
like .
scallops , and these were a-m-a-z-i-n-g .
fun and tasty cocktails .
next time i 'm in phoenix , i will go
back here .
highly recommend .

GT: 0 Prediction: 0

terrible value .
ordered pasta entree .
.
\$ 16.95 good taste but size was an
appetizer size .
.
no salad , no bread no vegetable .
this was .
our and tasty cocktails .
our second visit .
i will not go back .

Sentiment analysis

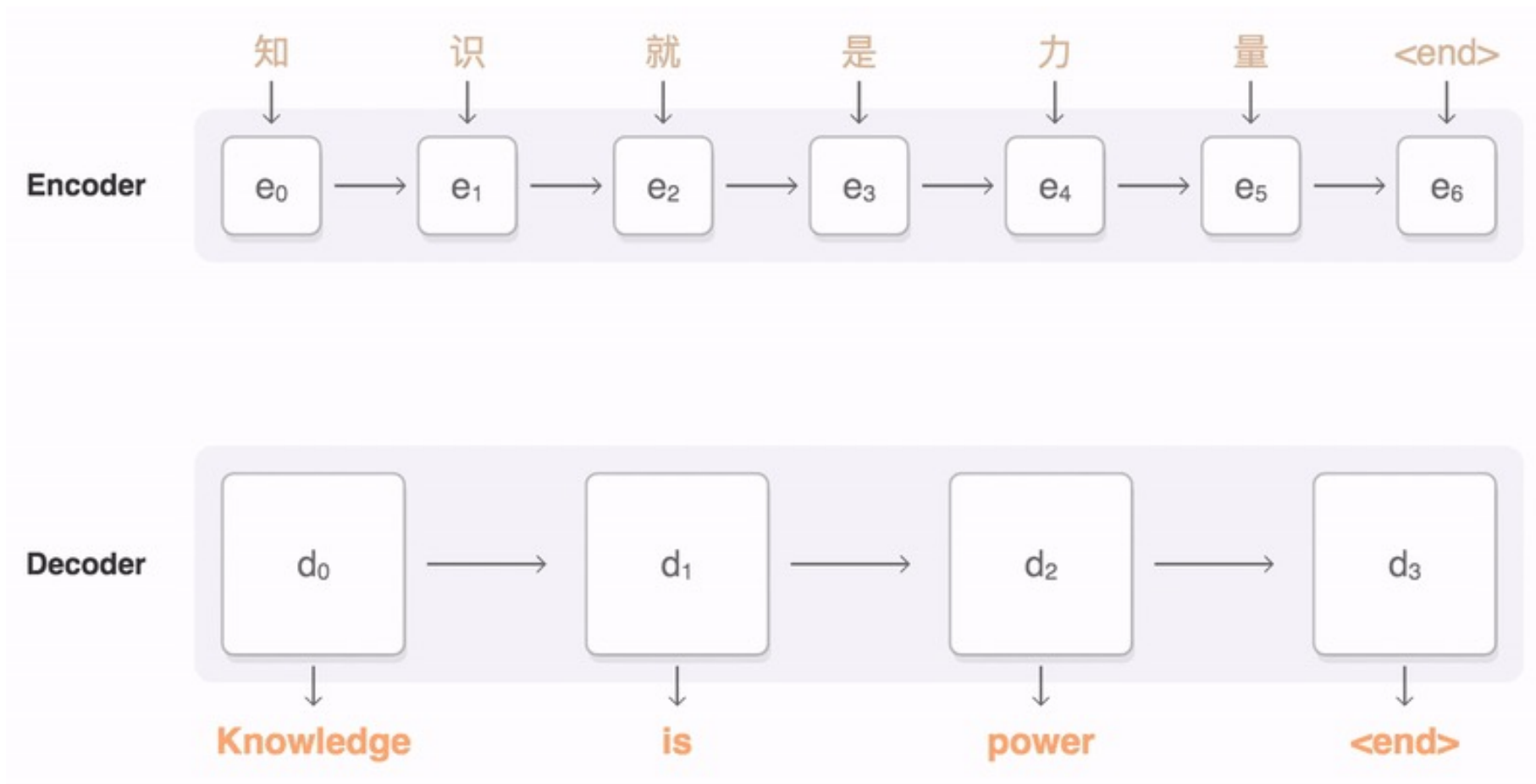
RNN Applications



Stock price prediction



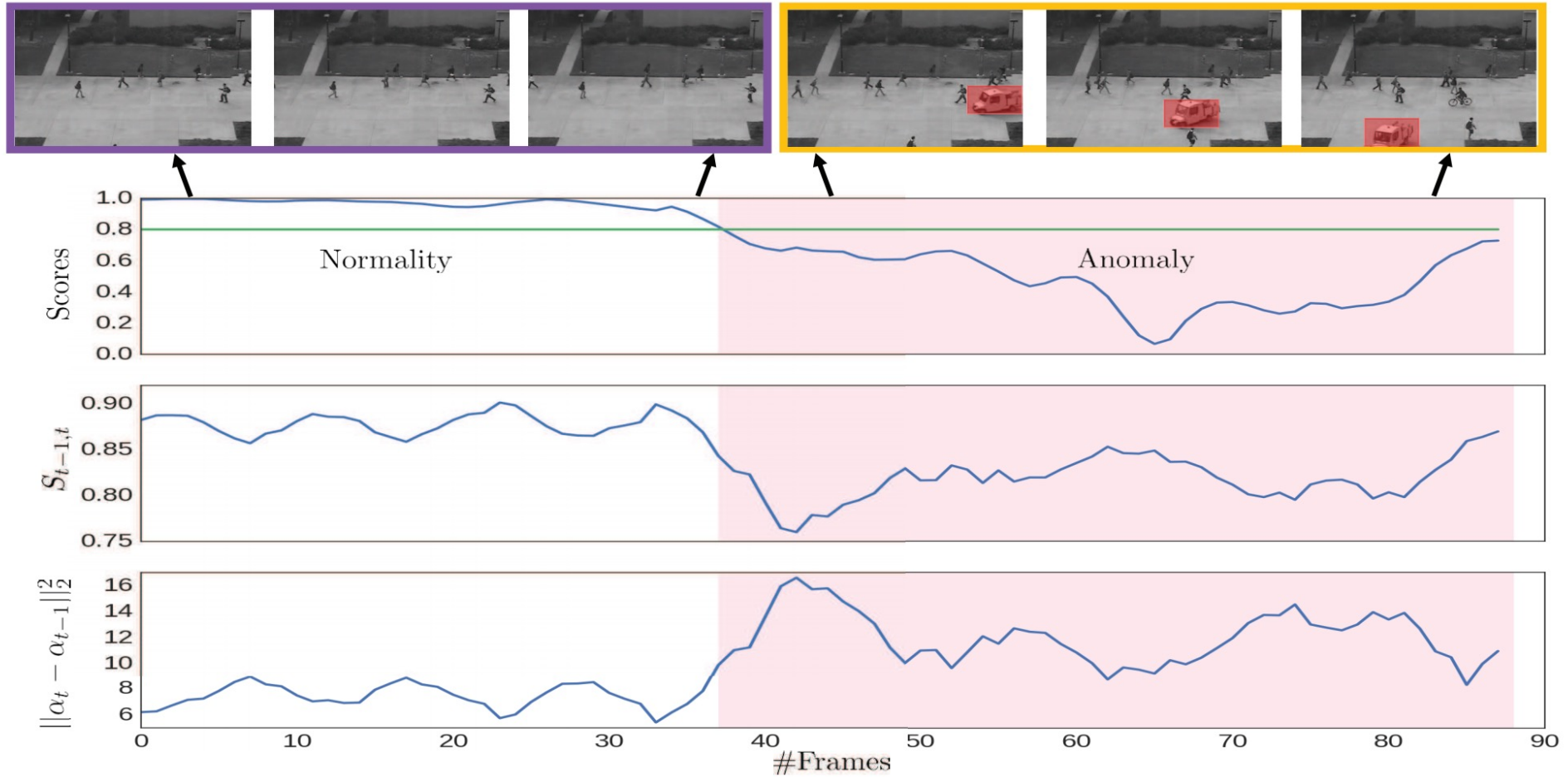
RNN Applications



Machine translation by seq2seq model



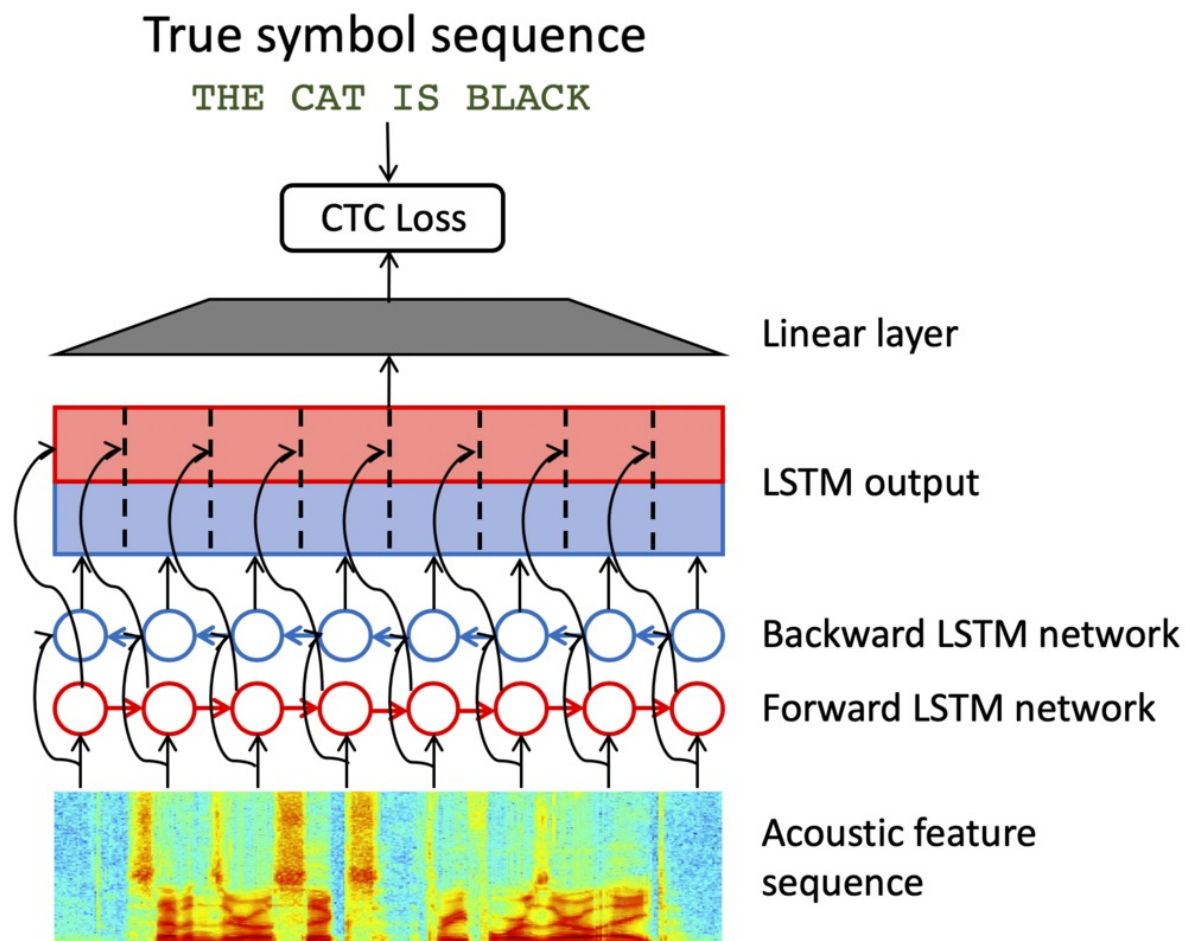
RNN Applications



Anomaly detection

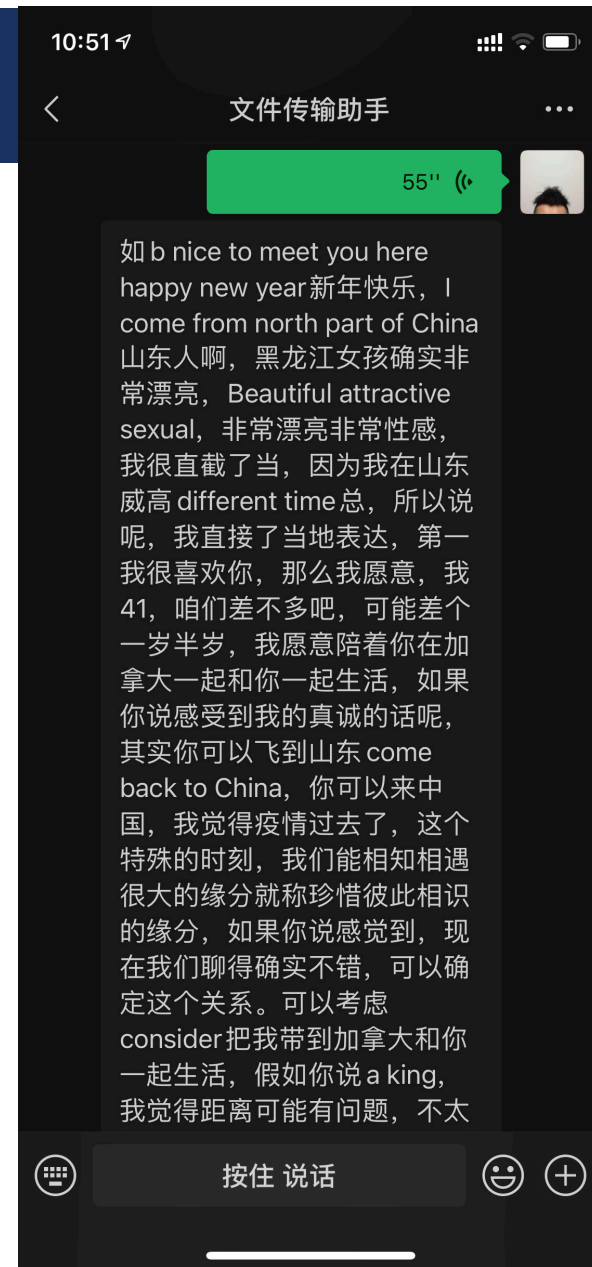


RNN Applications



Speech recognition

WeChat speech recognition of “山东king” 60s challenge



RNN Applications

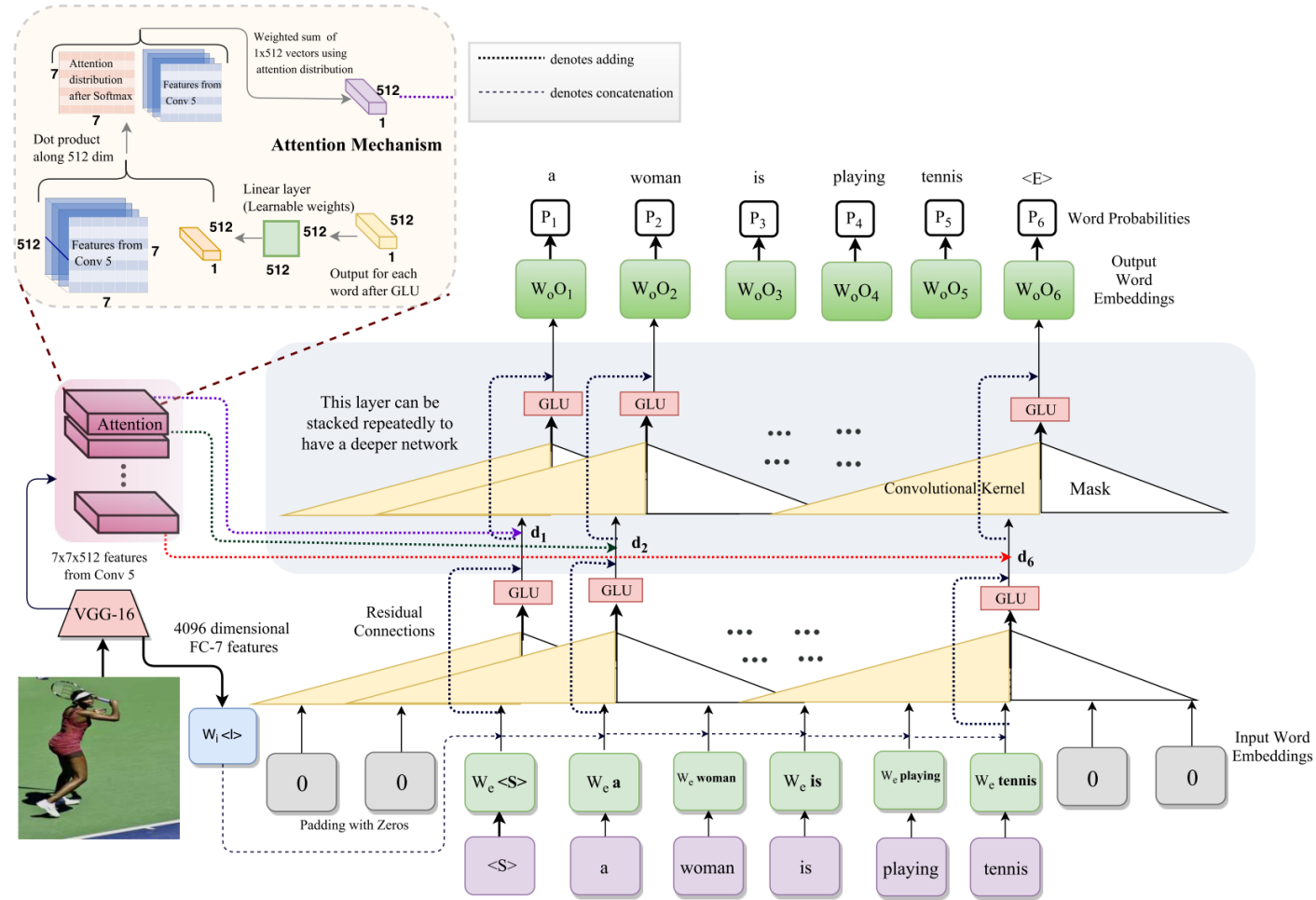


Image Captioning

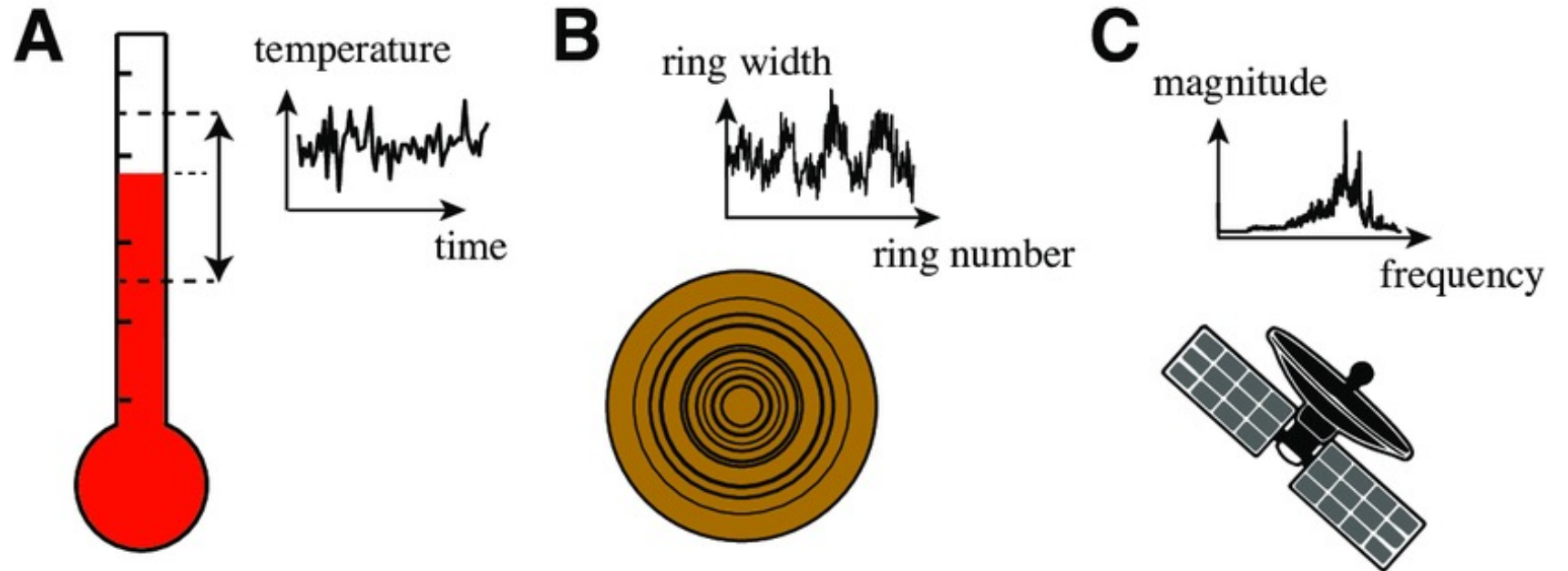


Sequential Data

- The data is ordered as a sequence.
- There are correlations between data in the sequence.

- Example:

- Speech.
- Text.
- Weather.
- User behavior.
- ...



Tasks with Sequential Data

Task	Example of Input	Example of Output
Speech recognition	Audio recording	Text in English
Language translation	Text in English	Text in Chinese
Image captioning	Image pixels	Short description in English
Sentiment classification	Product review	Positivity score
Stock price prediction	Historical stock prices	Future stock price
	Sequential data	Non-sequential data



Sequential Data

- Usually, we represent each sequential data sample as

$$\mathbf{x} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(\tau)}]$$

where the index t is the time step, totally we have τ time steps, and each $\mathbf{x}^{(t)}$ is a d -dim vector.

- $\mathbf{x}^{(t)}$ depends on $\mathbf{x}^{(t-1)}$.
- We can treat the sequential data sample as a whole ($d\tau$ -dim vector) and feed it into a MLP.
 - What are the problems here?

Sequential Data

Problems of training sequential data with MLP:

- In any sequence, **nearby items are related** and **the order of items matters**. Fully connected layer treats every item independently, until it learns otherwise.
- **Sequence length can vary** across examples within the same task. For MLP, shape of input and output is **determined** at design time.



RNN

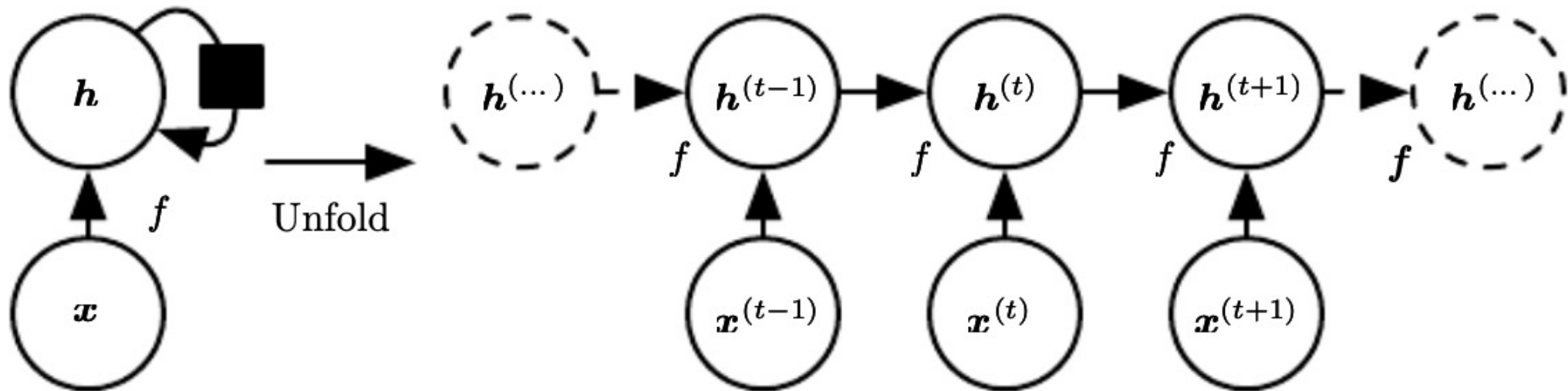
RNN

- A typical Recurrent Neural Network (RNN) uses the following equation:

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

where $\mathbf{h}^{(t)}$ is the hidden layer at time step t .

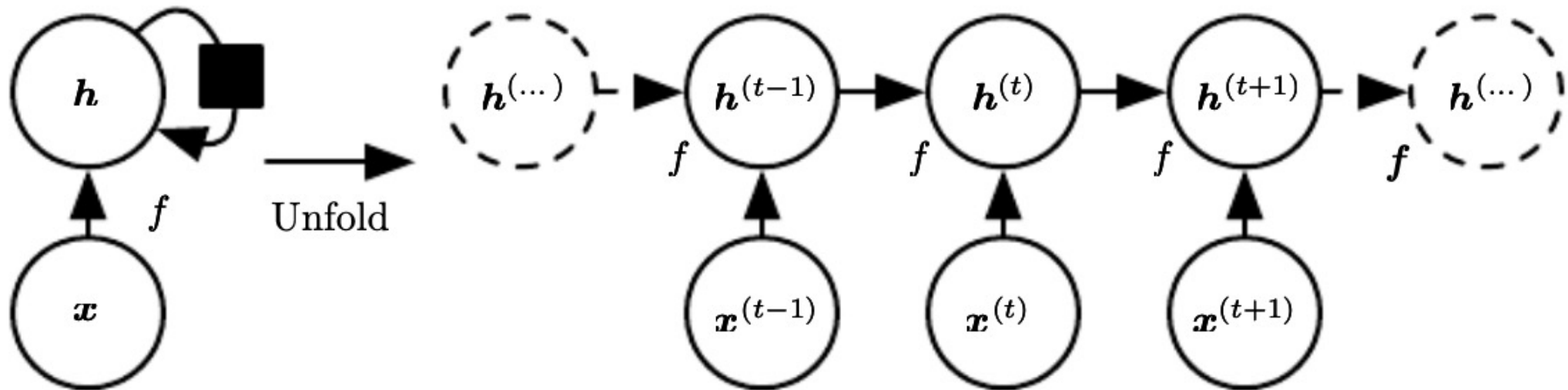
- The network typically learns to use $\mathbf{h}^{(t)}$ as a kind of **lossy summary** of the task-relevant aspects of the past sequence of inputs up to t .



RNN

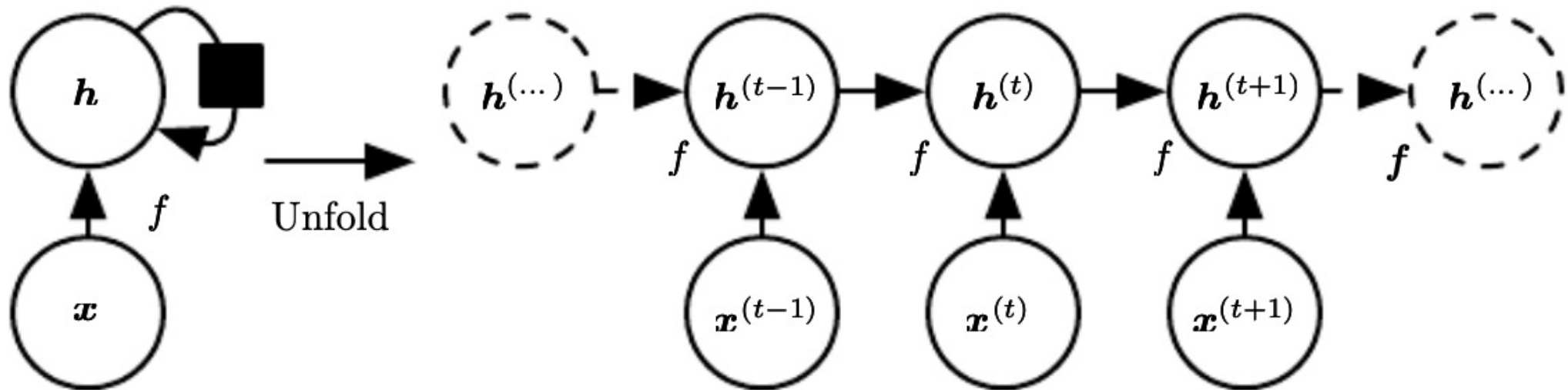
Two major advantages:

- Regardless of the sequence length, the learned model always has the **same input size**.
- It is possible to use the same transition function f with **the same parameters** at every time step.



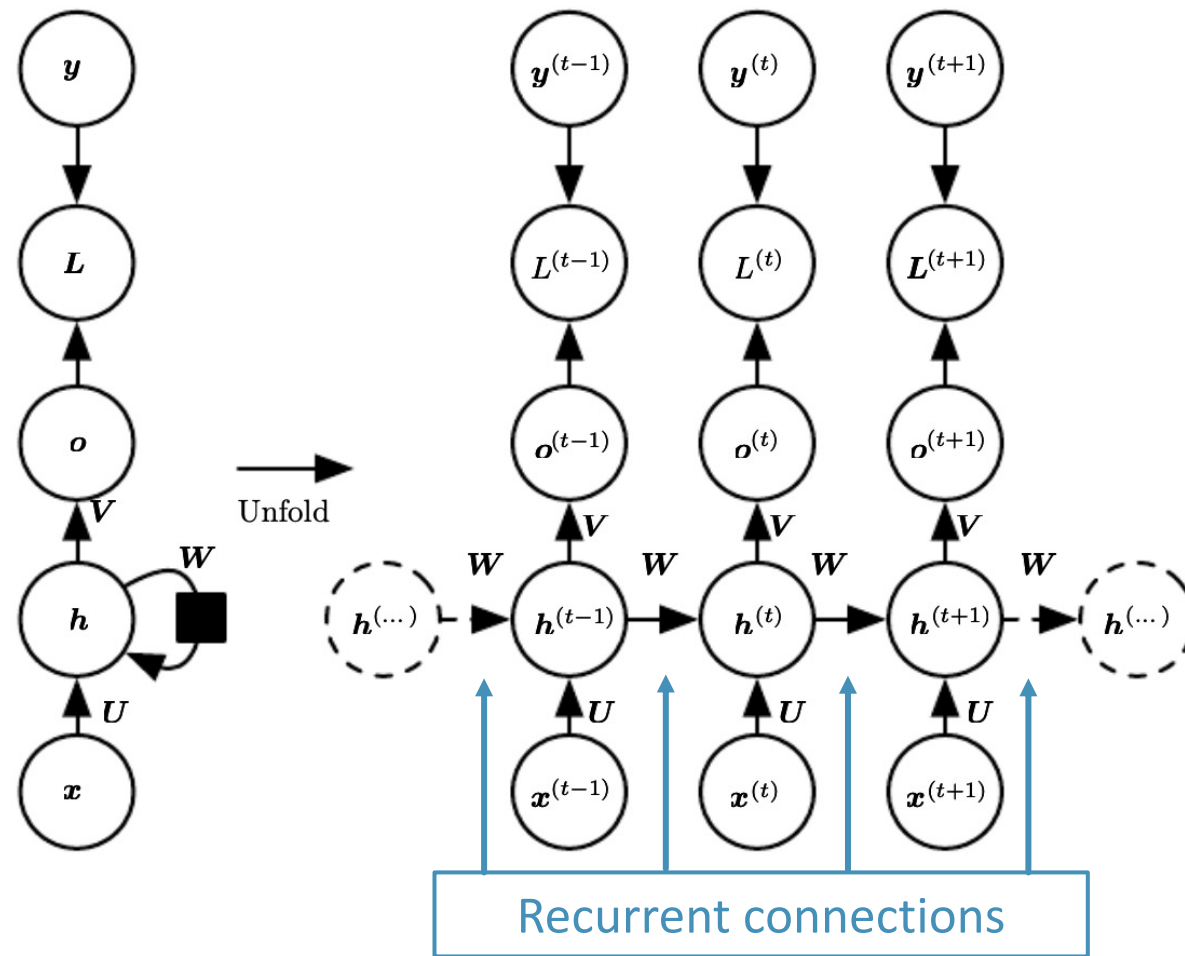
RNN

- It follows the same idea of **parameter sharing** as CNN.
- It is possible to learn **a single model f** that operates on all time steps and all sequence lengths, rather than needing to learn a separate model for all possible time steps.
- Learning a single, shared model allows **generalization** to sequence lengths that did not appear in the training set.



RNN

- U : a weight matrix for **input-to-hidden** connections.
- W : a weight matrix for **hidden-to-hidden recurrent connections**.
- V : a weight matrix for **hidden-to-output** connections.
- $o^{(t)}$: the corresponding output of input $x^{(t)}$.
- $L^{(t)}$: the loss measures how far each $o^{(t)}$ is from the corresponding training target $y^{(t)}$.



Vanilla RNN



RNN

- Forward propagation begins with a specification of the initial state $\mathbf{h}^{(0)}$.
- For each time step from $t = 1$ to $t = \tau$, we apply the following update equations:

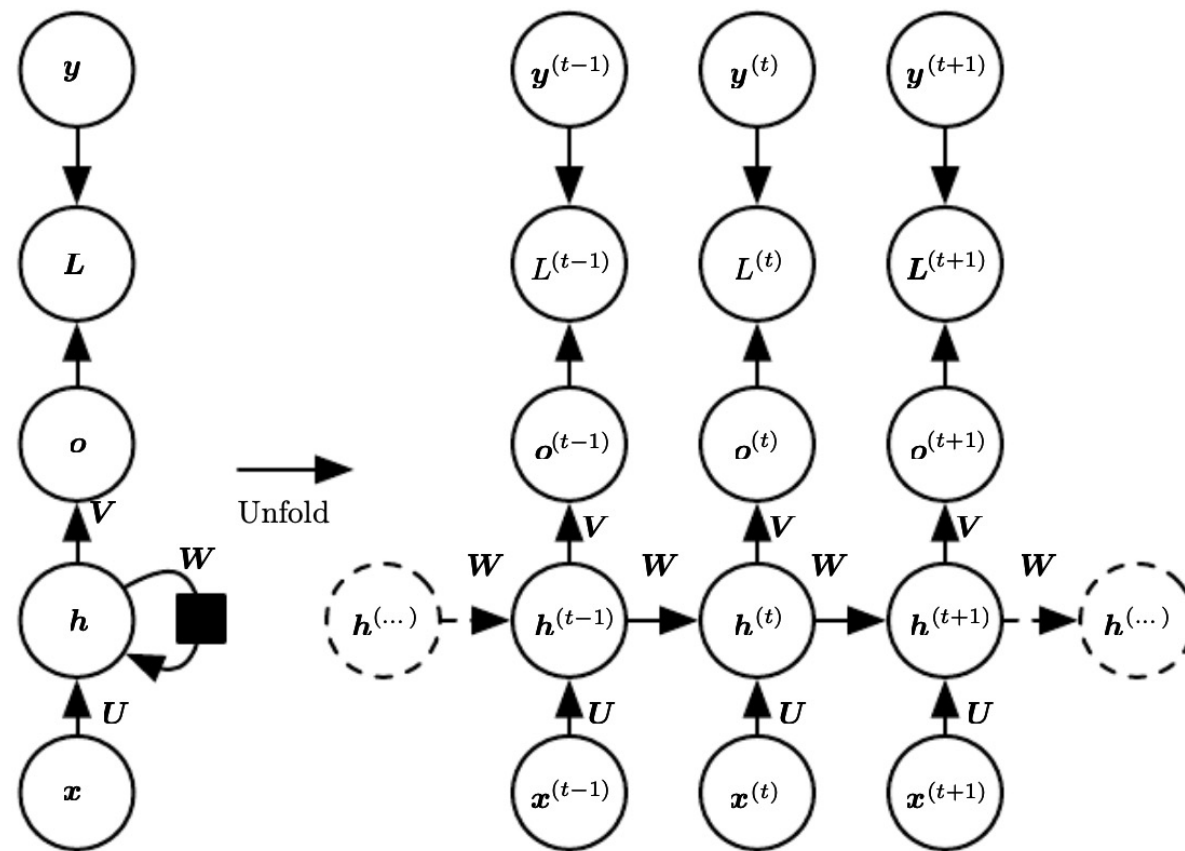
$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)},$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}),$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)},$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)}),$$

$$L^{(t)} = J(\hat{\mathbf{y}}^{(t)}, \mathbf{y}^{(t)}).$$



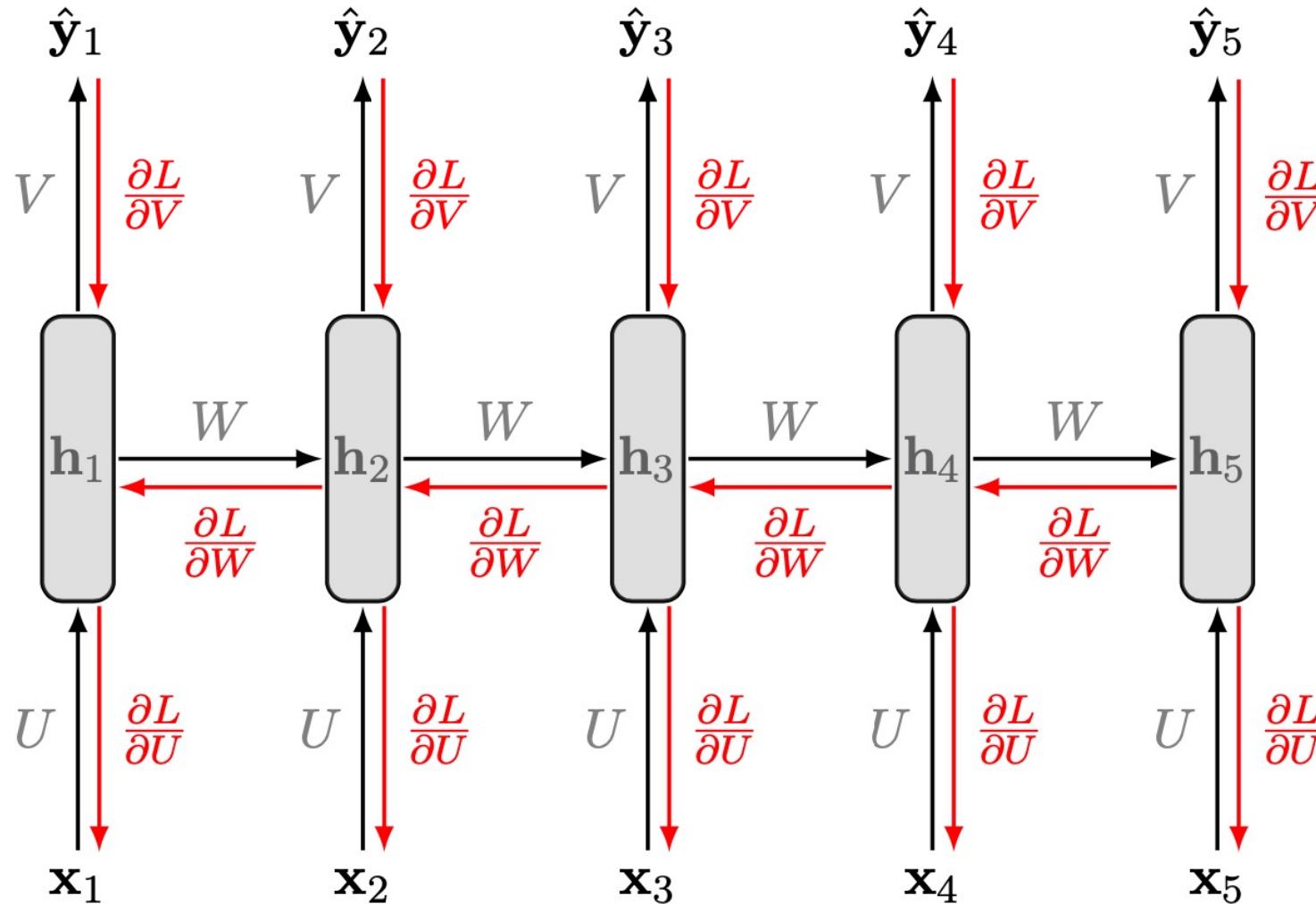
Vanilla RNN



Backpropagation Through Time

- Need to calculate the gradients for weights $\nabla_V L$, $\nabla_W L$, $\nabla_U L$, and biases $\nabla_c L$ and $\nabla_b L$
- Treat the recurrent network as a usual multilayer network and apply backpropagation on the unfold network.
- We move from the right to left: This is called **Backpropagation Through Time (BPTT)**.

Backpropagation Through Time



Backpropagation Through Time

$$\nabla_c L = \sum_t \nabla_{o^{(t)}} L,$$

$$\nabla_b L = \sum_t \text{diag} \left(1 - (\mathbf{h}^{(t)})^2 \right) \nabla_{\mathbf{h}^{(t)}} L,$$

$$\nabla_V L = \sum_t (\nabla_{o^{(t)}} L) \mathbf{h}^{(t)T},$$

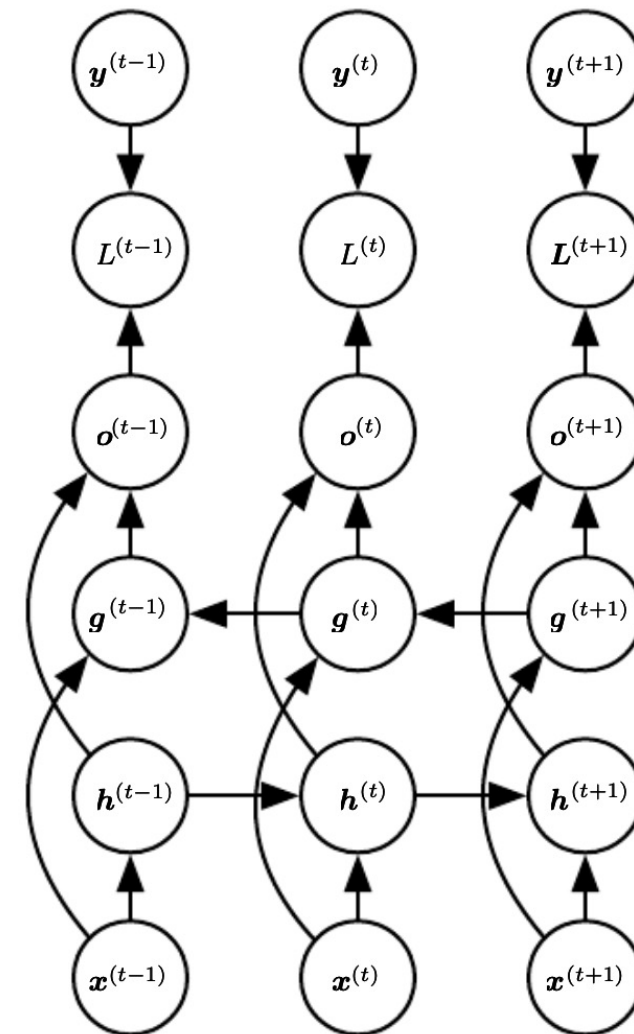
$$\nabla_W L = \sum_t \text{diag} \left(1 - (\mathbf{h}^{(t)})^2 \right) (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{h}^{(t-1)T},$$

$$\nabla_U L = \sum_t \text{diag} \left(1 - (\mathbf{h}^{(t)})^2 \right) (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{x}^{(t-1)T},$$



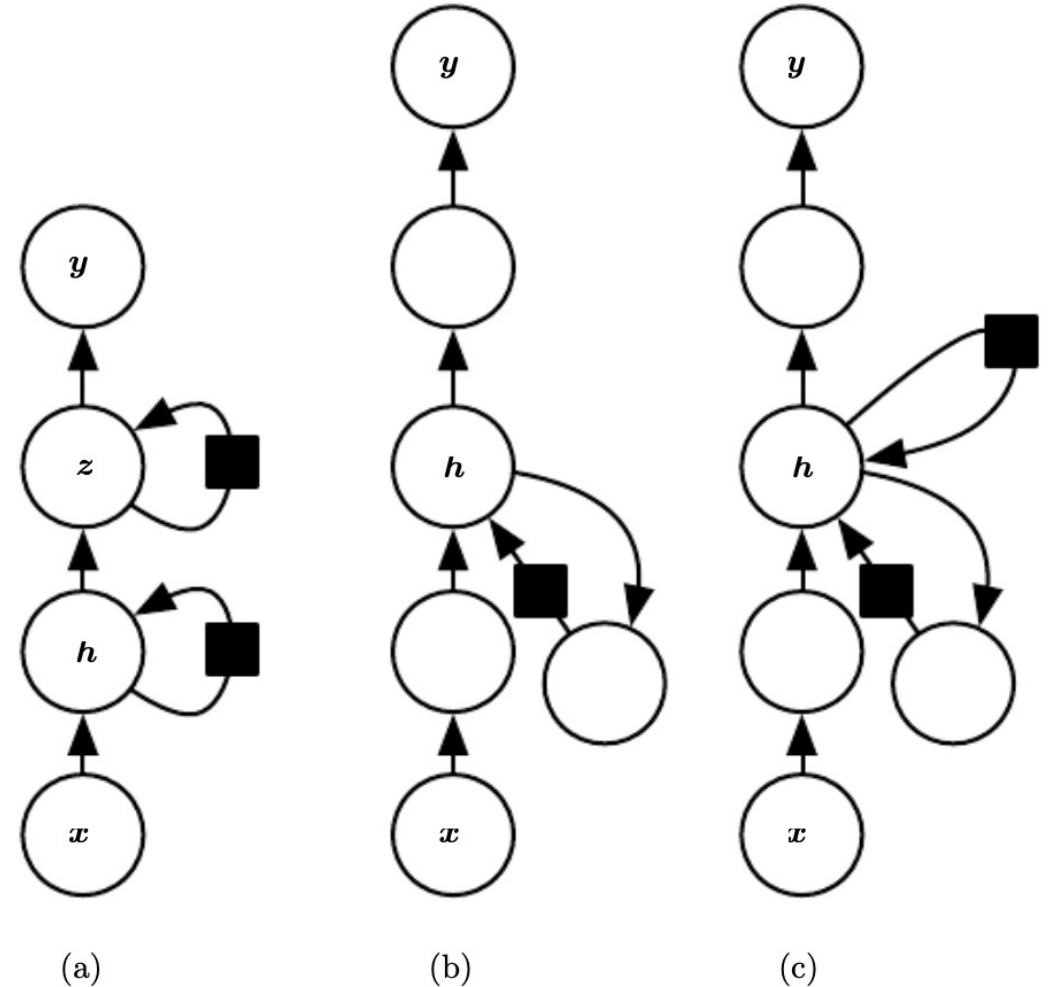
Bidirectional RNNs

- Up to now, the state at time t only captures information from the past $x^{(1)}, x^{(2)}, \dots, x^{(t-1)}$.
- However, in many applications we want to output a prediction of $y(t)$ which may **depend on the whole input sequence**.
- **Bidirectional RNNs** combine an RNN that moves **forward** with another RNN that moves **backward**.

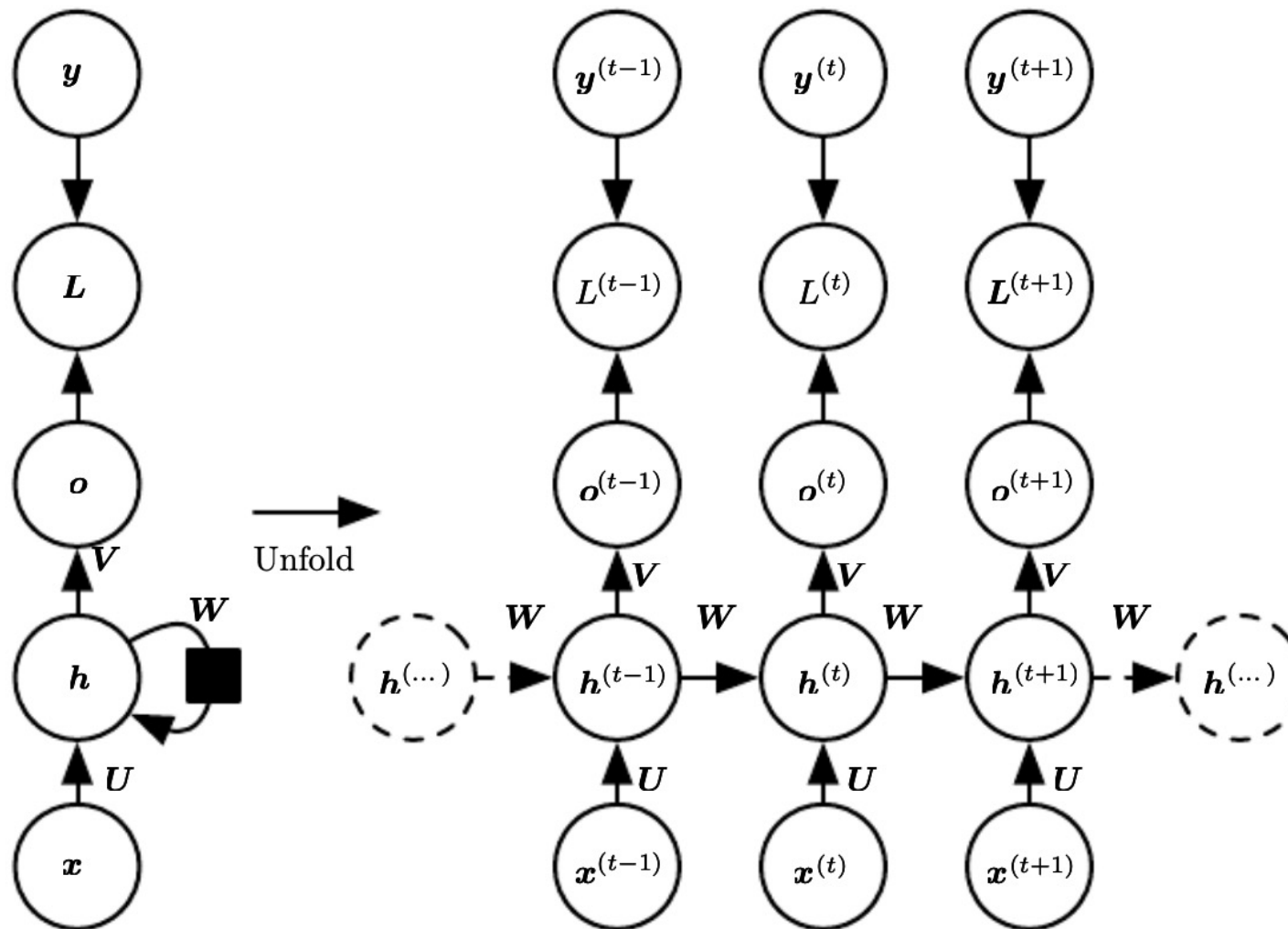


Deep Recurrent Networks

- The computation in most RNNs can be decomposed into three blocks of parameters and associated transformations:
 1. input to hidden;
 2. hidden to hidden;
 3. hidden to output.
- Would it be advantageous to introduce depth in each of these operations?
 - Of course, and the deep module can be incorporated into different RNN components.

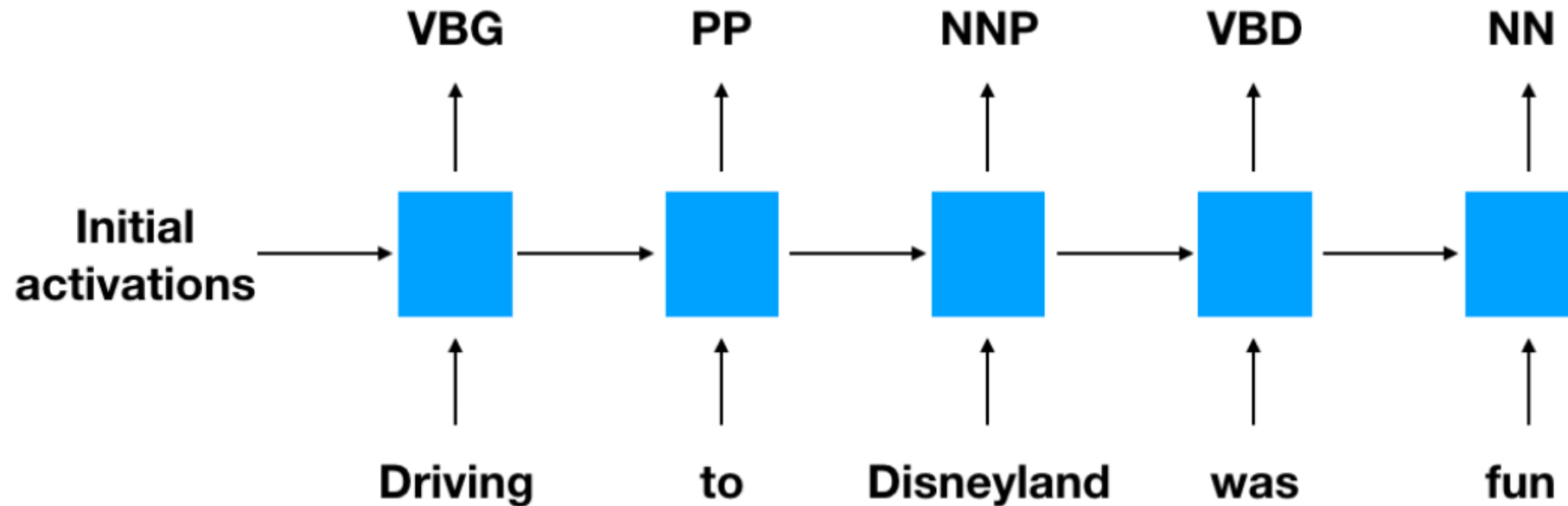


RNN: Many-to-Many

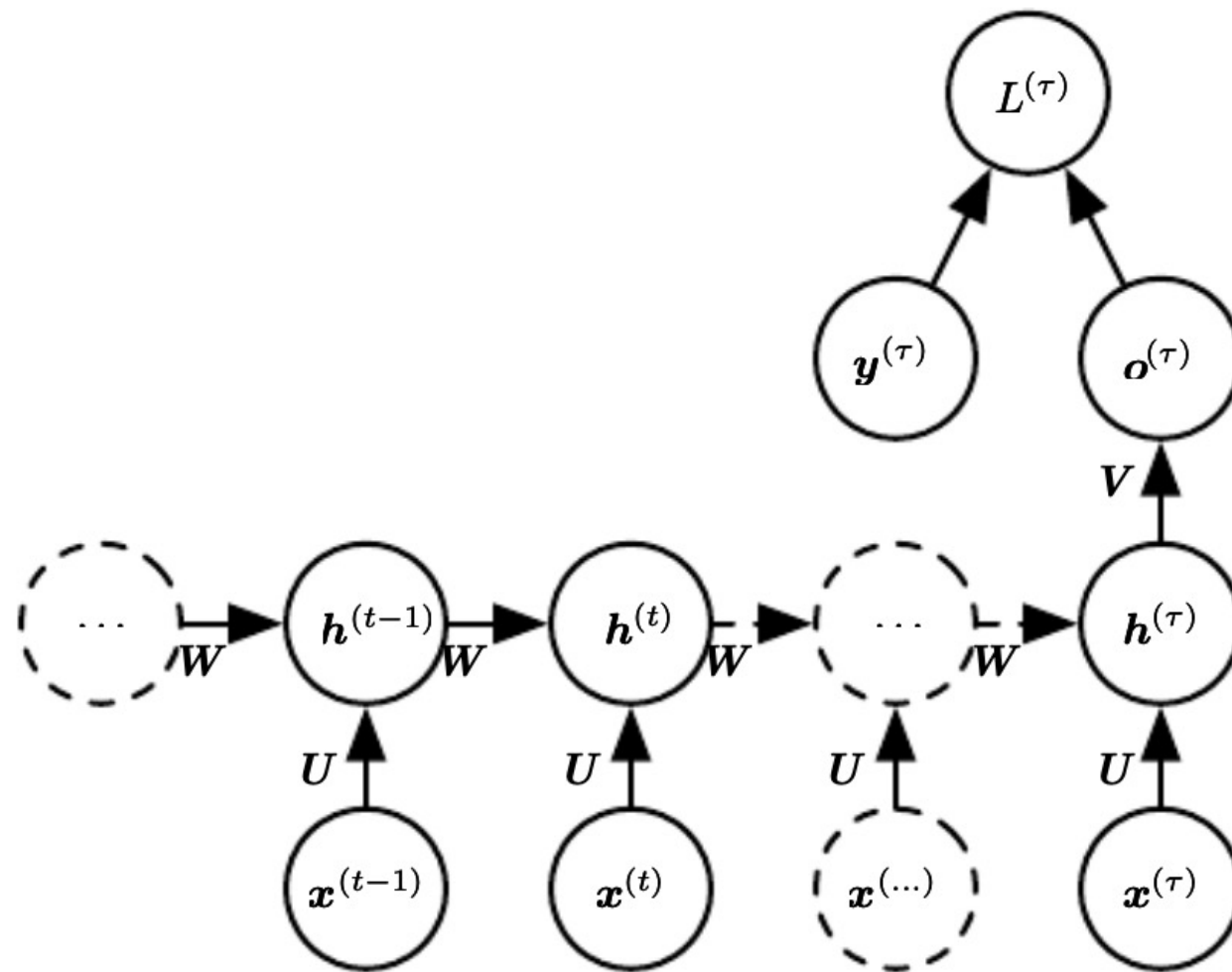


RNN: Many-to-Many

- Application: Part-of-Speech (POS) tagging

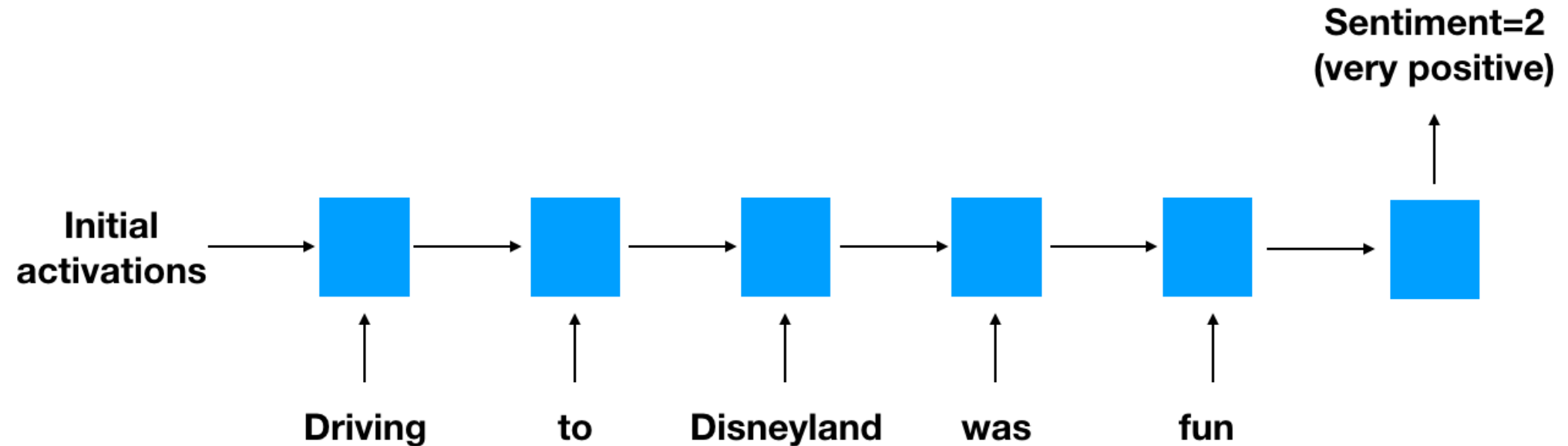


RNN: Many-to-One

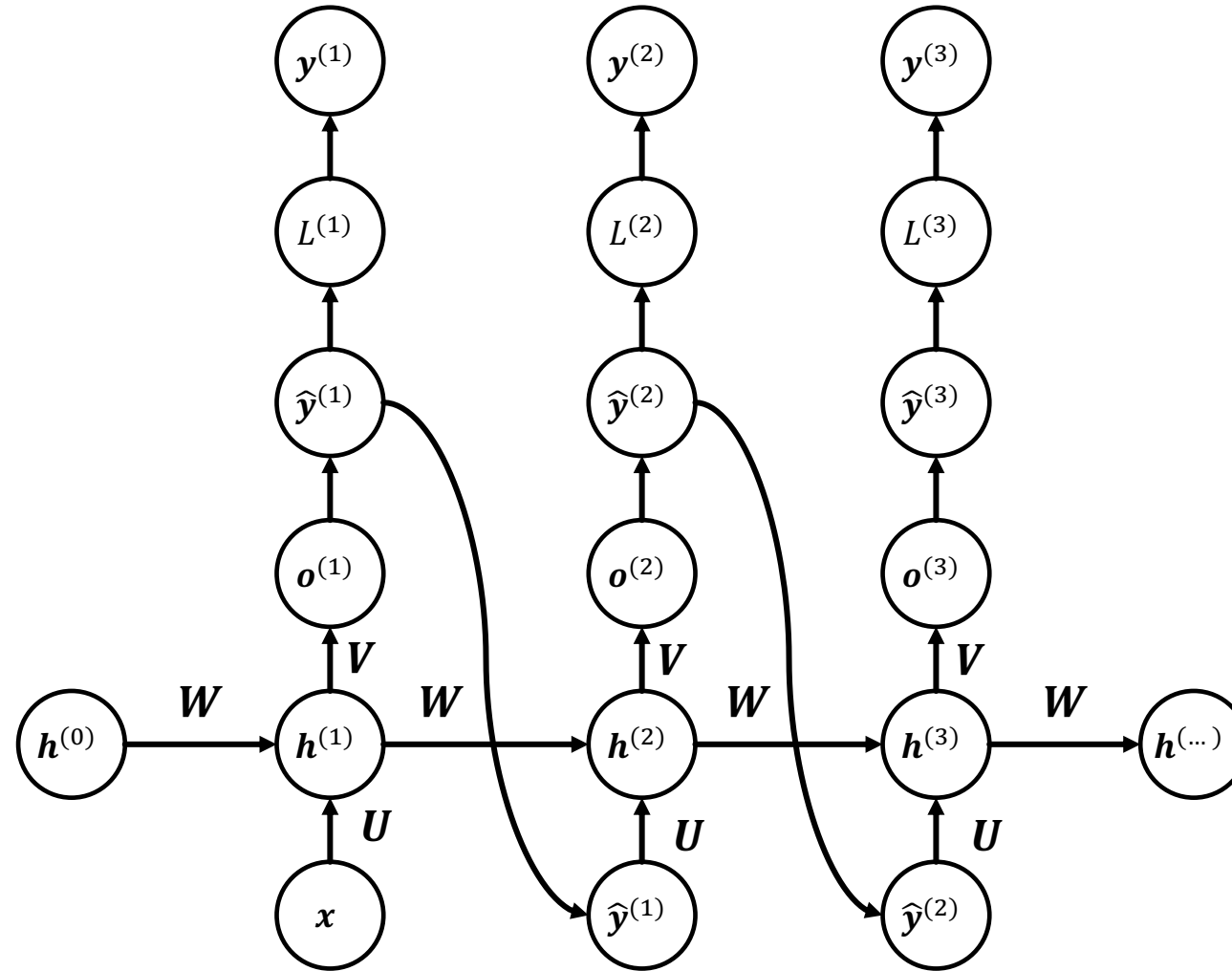


RNN: Many-to-One

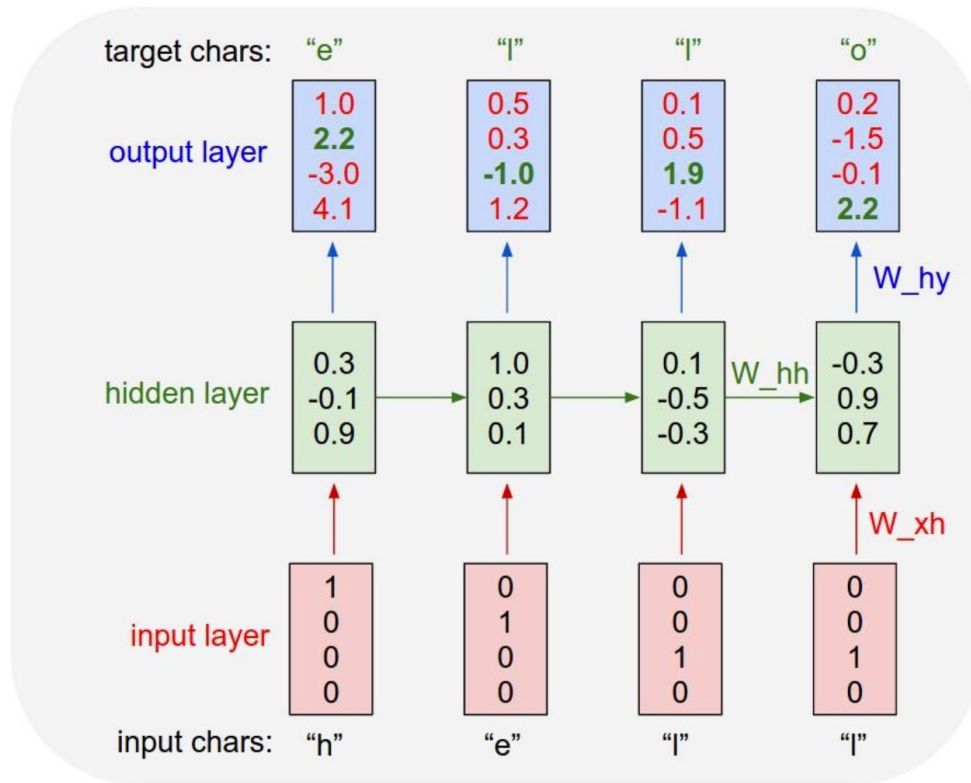
■ Application: Sentiment analysis



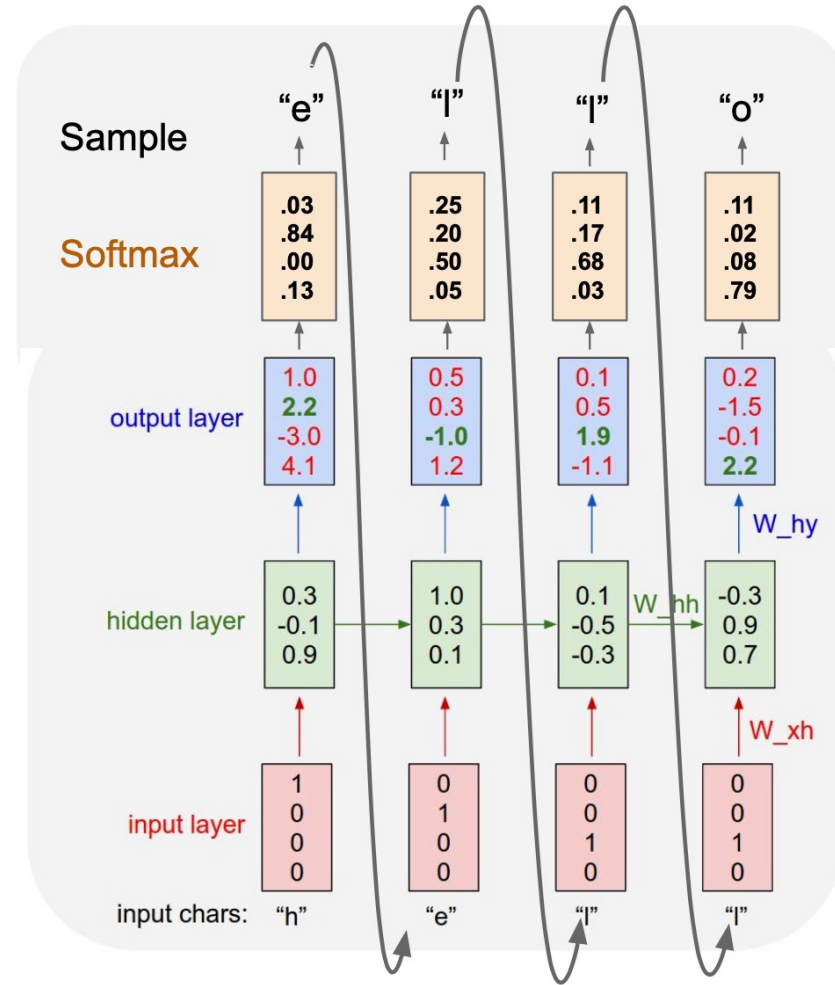
RNN: One-to-Many



RNN: One-to-Many



Training phase (many-to-many)



Test phase (one-to-many)

RNN: One-to-Many

- The RNN takes a word, the context from previous time steps and defines a distribution over the next word in the sentence.
- The RNN is conditioned on the image information at the first time step.
- “START” and “END” are special tokens.

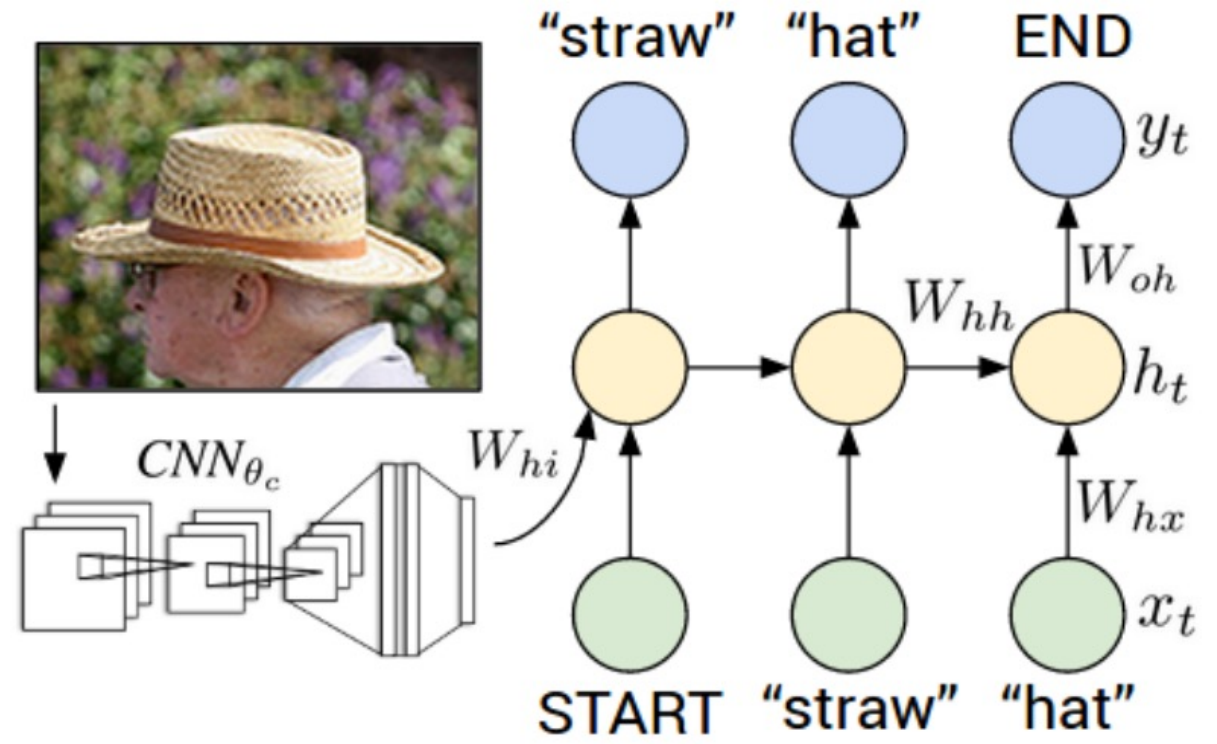


Image captioning

Example

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

VIOLA:

Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:

O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

Generated Shakespear work by RNN. It was trained on 4.4MB file containing all Shakespeare work.



Example

Proof. Omitted. □

Lemma 0.1. *Let \mathcal{C} be a set of the construction.*

Let \mathcal{C} be a gerber covering. Let \mathcal{F} be a quasi-coherent sheaves of \mathcal{O} -modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

Proof. This is an algebraic space with the composition of sheaves \mathcal{F} on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X}(\mathcal{G}, \mathcal{F})\}$$

where \mathcal{G} defines an isomorphism $\mathcal{F} \rightarrow \mathcal{F}$ of \mathcal{O} -modules. □

Lemma 0.2. *This is an integer Z is injective.*

Proof. See Spaces, Lemma ?? □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset X$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \rightarrow Y' \rightarrow Y \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

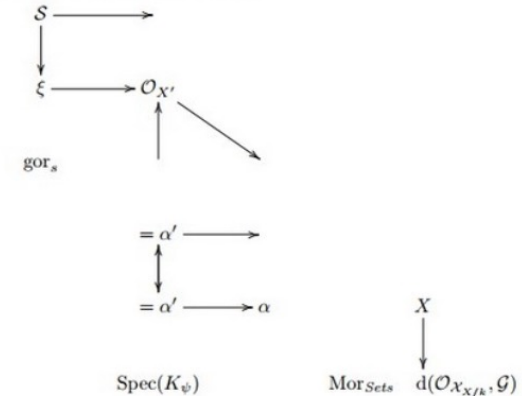
be a morphism of algebraic spaces over S and Y .

Proof. Let X be a nonzero scheme of X . Let X be an algebraic space. Let \mathcal{F} be a quasi-coherent sheaf of \mathcal{O}_X -modules. The following are equivalent

- (1) \mathcal{F} is an algebraic space over S .
- (2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then \mathcal{G} is a finite type and assume S is a flat and \mathcal{F} and \mathcal{G} is a finite type f_* . This is of finite type diagrams, and

- the composition of \mathcal{G} is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \text{Spec}(R)$ and \mathcal{F} is a finite type representable by algebraic space. The property \mathcal{F} is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U . □

Proof. This is clear that \mathcal{G} is a finite presentation, see Lemmas ??.

A reduced above we conclude that U is an open covering of \mathcal{C} . The functor \mathcal{F} is a "field

$$\mathcal{O}_{X,x} \rightarrow \mathcal{F}_{\bar{x}}^{-1}(\mathcal{O}_{X_{\acute{e}tale}}) \rightarrow \mathcal{O}_{X_{\acute{e}}}^{-1} \mathcal{O}_{X_{\lambda}}(\mathcal{O}_{X_n}^{\bar{v}})$$

is an isomorphism of covering of \mathcal{O}_{X_i} . If \mathcal{F} is the unique element of \mathcal{F} such that X is an isomorphism.

The property \mathcal{F} is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme \mathcal{O}_X -algebra with \mathcal{F} are opens of finite type over S .

If \mathcal{F} is a scheme theoretic image points. □

If \mathcal{F} is a finite direct sum $\mathcal{O}_{X_{\lambda}}$ is a closed immersion, see Lemma ?? . This is a sequence of \mathcal{F} is a similar morphism.

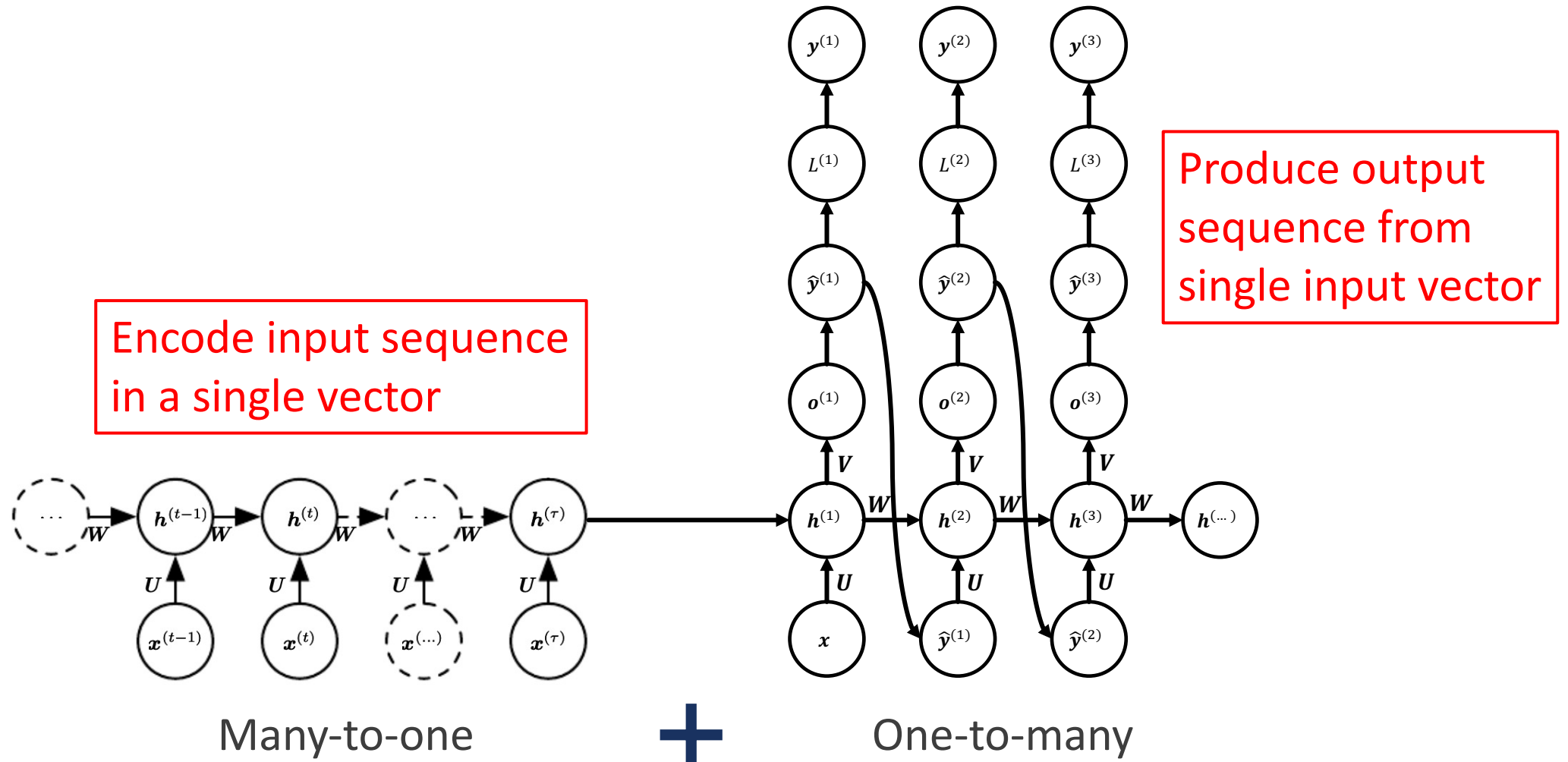
Generated LaTeX file by RNN. It is trained on 16MB LaTeX source file on algebraic geometry.





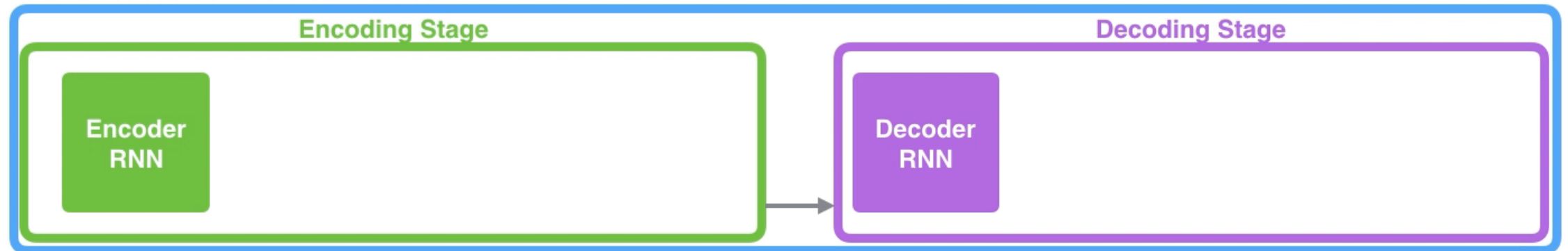
SEQ2SEQ AND ATTENTION MODELS

Sequence to Sequence



Sequence to Sequence

Neural Machine Translation SEQUENCE TO SEQUENCE MODEL



Je

suis

étudiant

Encoder

Decoder



Sequence to Sequence

- Sequence to Sequence (seq2seq) is the key technique of Neural Machine Translation (NMT).
- The steps of a typical NMT model:
 - The input sentence $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)}$ via hidden unit activations $\mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \dots, \mathbf{h}^{(n)}$ is encoded into the thought vector \mathbf{c} .
 - Using \mathbf{c} , the decoder then generates the output sentence $\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \dots, \mathbf{y}^{(p)}$.
 - We stop when we sample a terminating token i.e. $\mathbf{y}^{(p)} = \langle END \rangle$.
- A Problem? For long sentences, it is not enough for the decoder if only the vector \mathbf{c} is given.



Attention Models

Neural machine translation by jointly learning to align and translate

[D Bahdanau, K Cho, Y Bengio](#) - arXiv preprint arXiv:1409.0473, 2014 - arxiv.org

... is a recently proposed approach to **machine translation**. Unlike the ... **machine translation**, the **neural machine translation** aims at building a single **neural** network that can be **jointly** tuned ...

☆ Save 77 Cite Cited by 31385 Related articles All 29 versions ⇔

- When we ourselves are translating a sentence from one language to another, we don't consider the whole sentence at all times.
- Intuition: Every word in the output only **depends on a word or a group of words in the input sentence**.
- We would like the decoder, while it is about to generate the next word, **to pay attention to only a group of words in the input sentence** most relevant to predicting the right next word.

Attention Models

- s_i is the hidden state of decoder RNN for time i :

$$s_i = f(s_{i-1}, c_i).$$

- c_i is the **context vector** computed as a weighted sum of the hidden state of encoder RNN h_j :

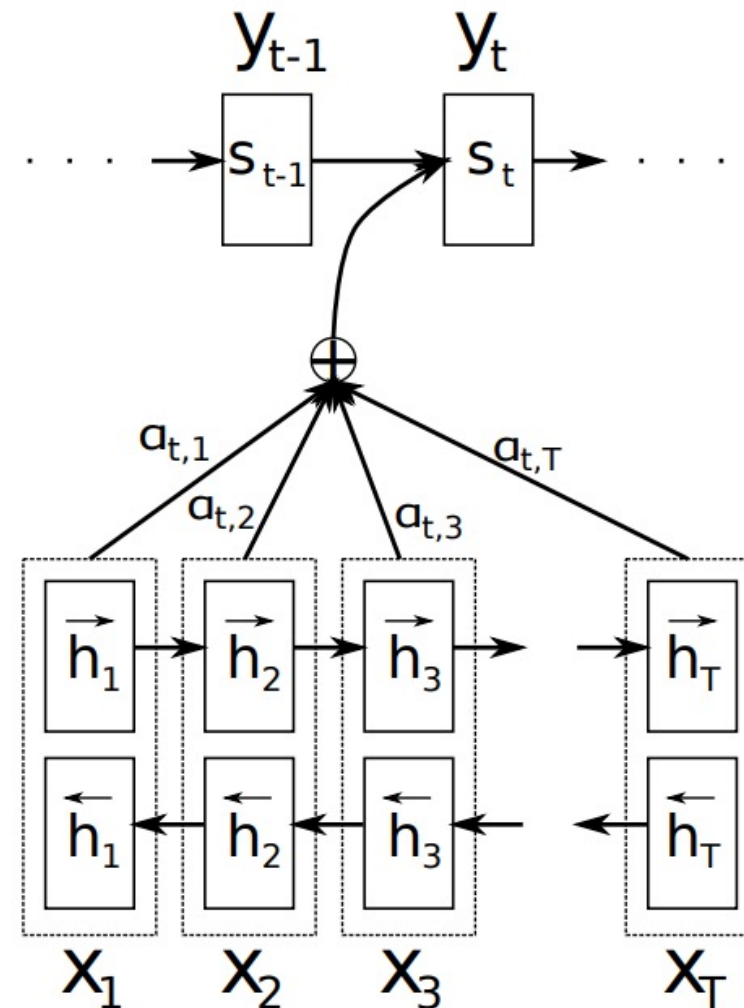
$$c_i = \sum_{j=1}^T \alpha_{ij} h_j.$$

- Each weight α_{ij} is calculated by:

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

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

The weight is determined by s_{i-1} and h_j



where a as a feedforward neural network which is jointly trained with all the other component.

Current translation status (特色) Original text

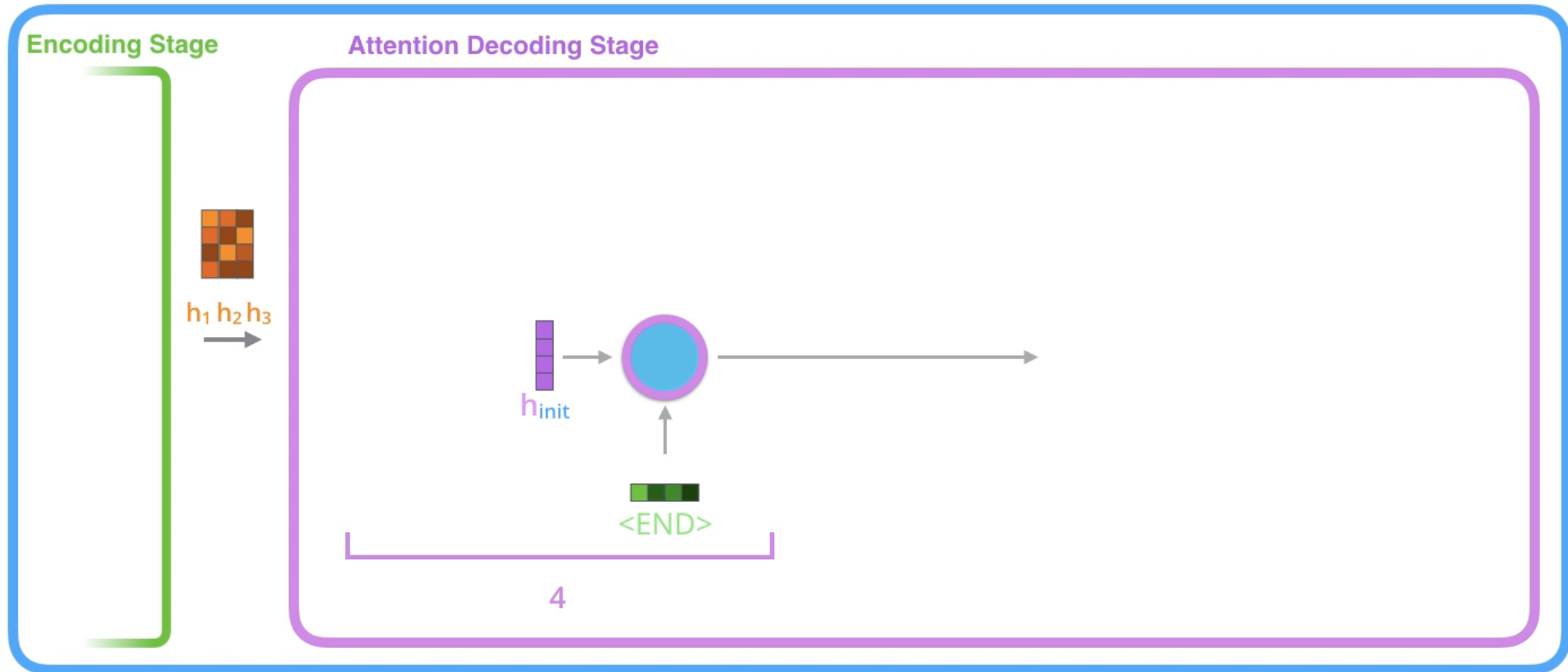
Attention Models

Attention at time step 4

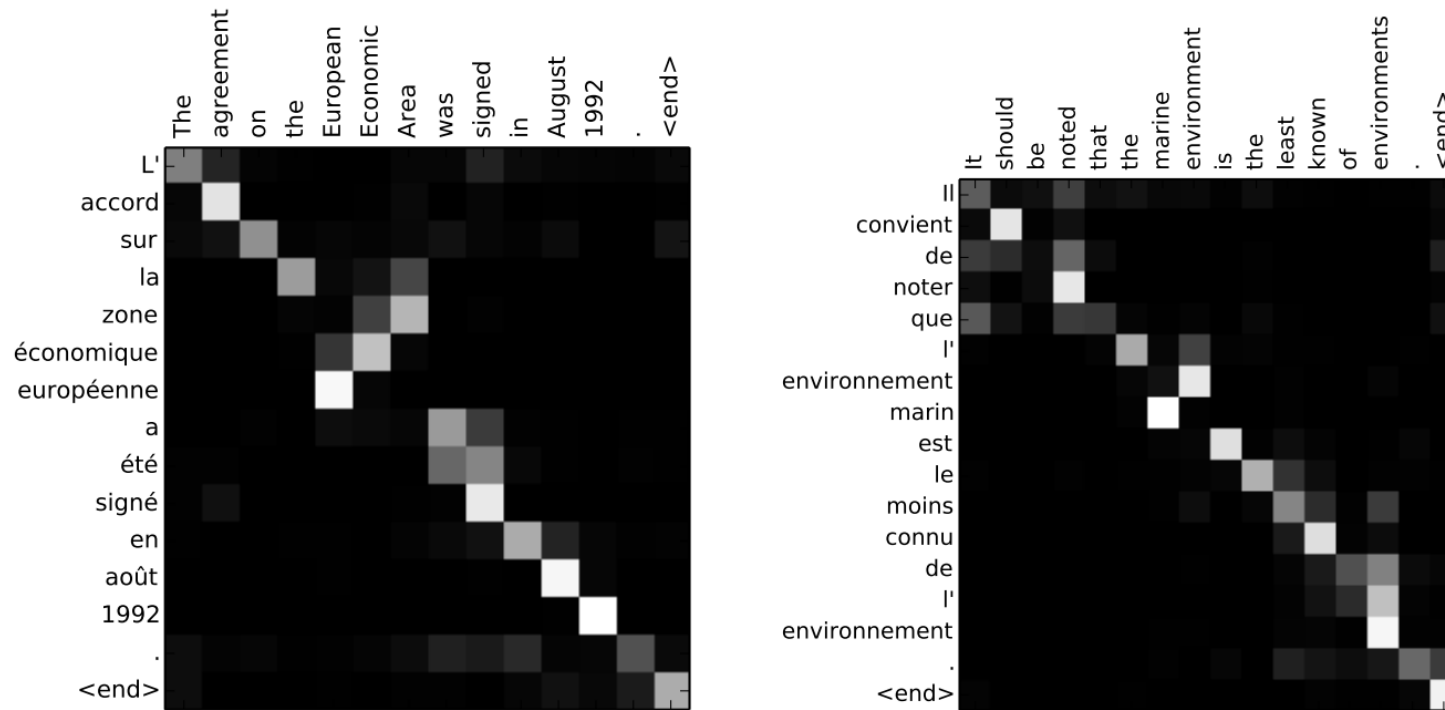


Attention Models

Neural Machine Translation SEQUENCE TO SEQUENCE MODEL WITH ATTENTION



Attention Models



- The matrix gives the degree of focus on all the input words.
- A linear order is not forced, but it figures out that **the translation is approximately linear**.



Attention in CV

Show, attend and tell: Neural image caption generation with visual attention

[K Xu, J Ba, R Kiros, K Cho, A Courville...](#) - International ..., 2015 - proceedings.mlr.press

... to the task at hand, **and** we **show** how learning to **attend** at different locations in order to ... attention mechanism **and** a “soft” deterministic attention mechanism. We also **show** how one ...

☆ Save ↗ Cite Cited by 11326 Related articles All 30 versions ⇨

- Attention can also be used to understand images.
- Humans don't process a visual scene all at once.
 - The Fovea gives high resolution vision in only a tiny region of our field of view.
- A series of glimpses are then integrated.

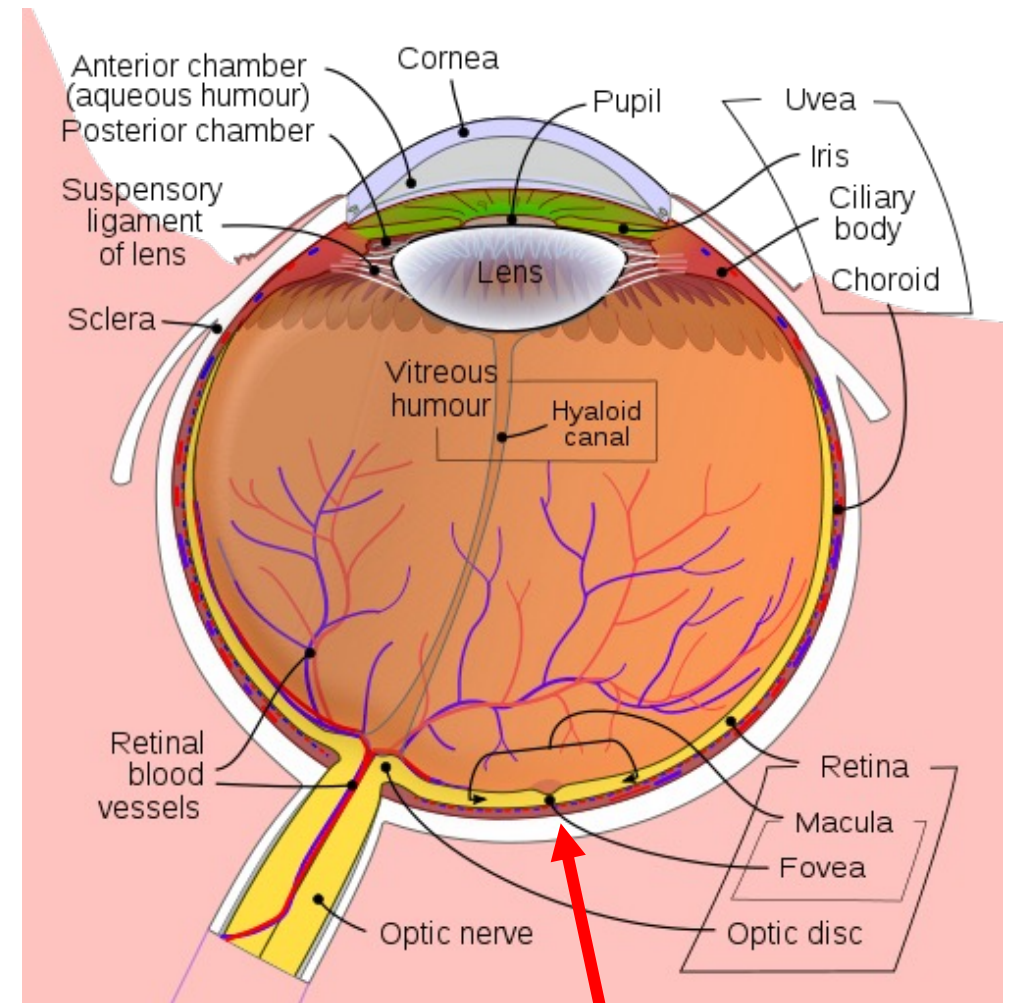


Image Captioning using Attention

- Here we have an encoder and decoder as well:
 - Encoder: A trained network like ResNet that extracts features for an input image.
 - Decoder: Attention based RNN, which is like the decoder in the translation model.
- While generating the caption, at every time step, the decoder must decide **which region of the image to focus**.

Image Captioning using Attention

- Not only generates good captions, but we also get to see where the decoder is looking at in the image.
- Attention also play important roles for model interpretation.

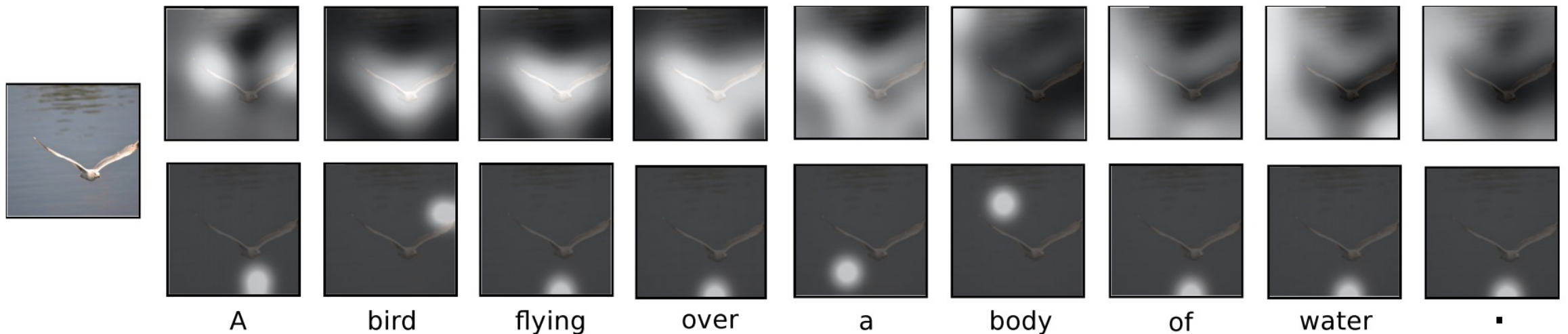
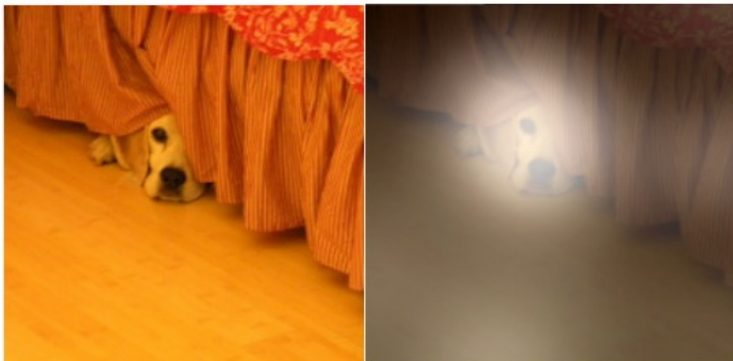


Image Captioning using Attention



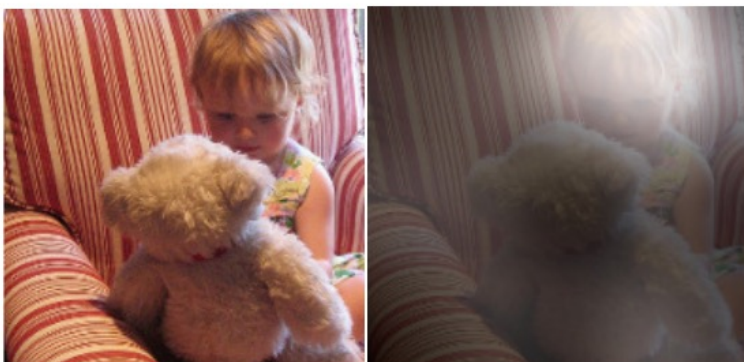
A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



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



A little girl sitting on a bed with a teddy bear.



A group of people sitting on a boat in the water.

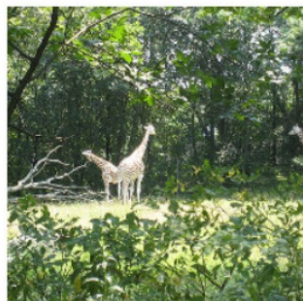


A giraffe standing in a forest with trees in the background.

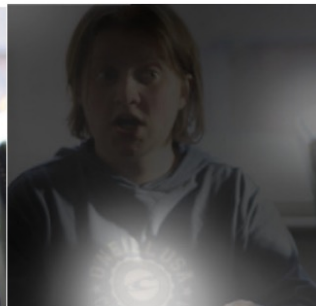
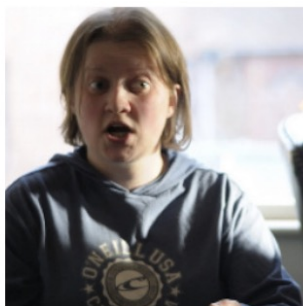


Image Captioning using Attention

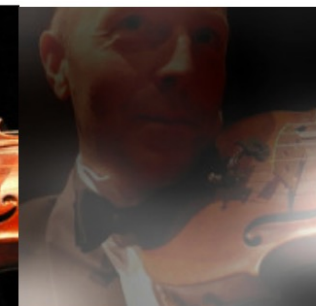
- Model interpretation is especially important when the prediction is wrong.



A large white bird standing in a forest.



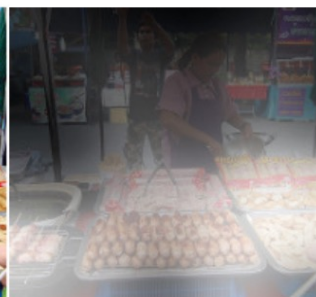
A woman holding a clock in her hand.



A man wearing a hat and a hat on a skateboard.



A person is standing on a beach with a surfboard.



A woman is sitting at a table with a large pizza.



A man is talking on his cell phone while another man watches.

Wrong captions

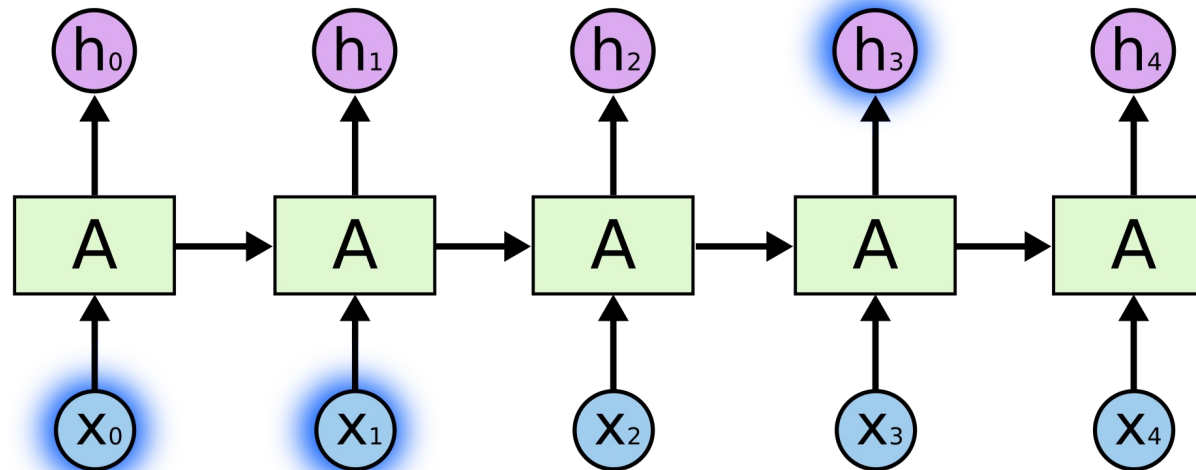




LSTM

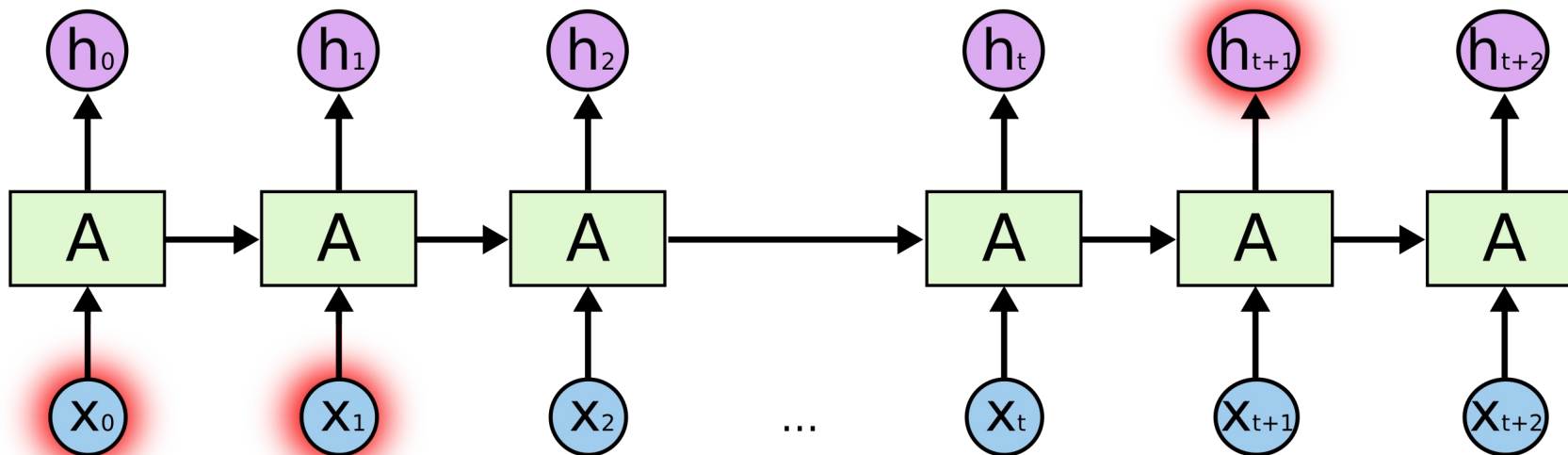
The Problem of Long-Term Dependencies

- The most appealing advantage of RNNs is the ability to **connect previous information to the present task**.
 - E.g. using previous video frames might inform the understanding of the present frame.
- If the gap is not very far, it seems ok.
 - E.g. the task of next word prediction: “the clouds are in the _”.



The Problem of Long-Term Dependencies

- Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.
 - E.g. “I grew up in China ... (300 words)... Of course I can speak fluent _.”
- In theory, RNNs are absolutely capable of handling such “long-term dependencies.” But in practice, RNNs have difficulties to learn them.



LSTMs

Long short-term memory

[S Hochreiter, J Schmidhuber - Neural computation, 1997 - ieeexplore.ieee.org](#)

... (**short-term memory**, as opposed to **long-term memory** ... learning what to put in **shortterm memory**, however, take too ... and corresponding teacher signals are **long**. Although theoretically ...

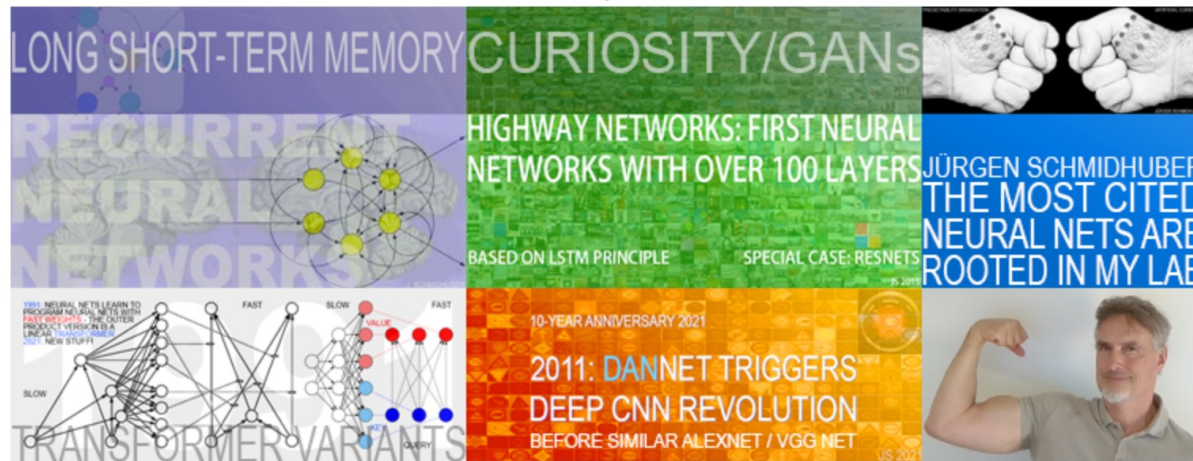
☆ Save 剪 Cite Cited by 92436 Related articles All 45 versions 》

- Long Short Term Memory networks (LSTMs) is a special kind of RNN, explicitly designed for learning long-term dependencies.
- It was proposed in 1997 by Jürgen Schmidhuber, but became popular until the deep learning era.



Jürgen Schmidhuber





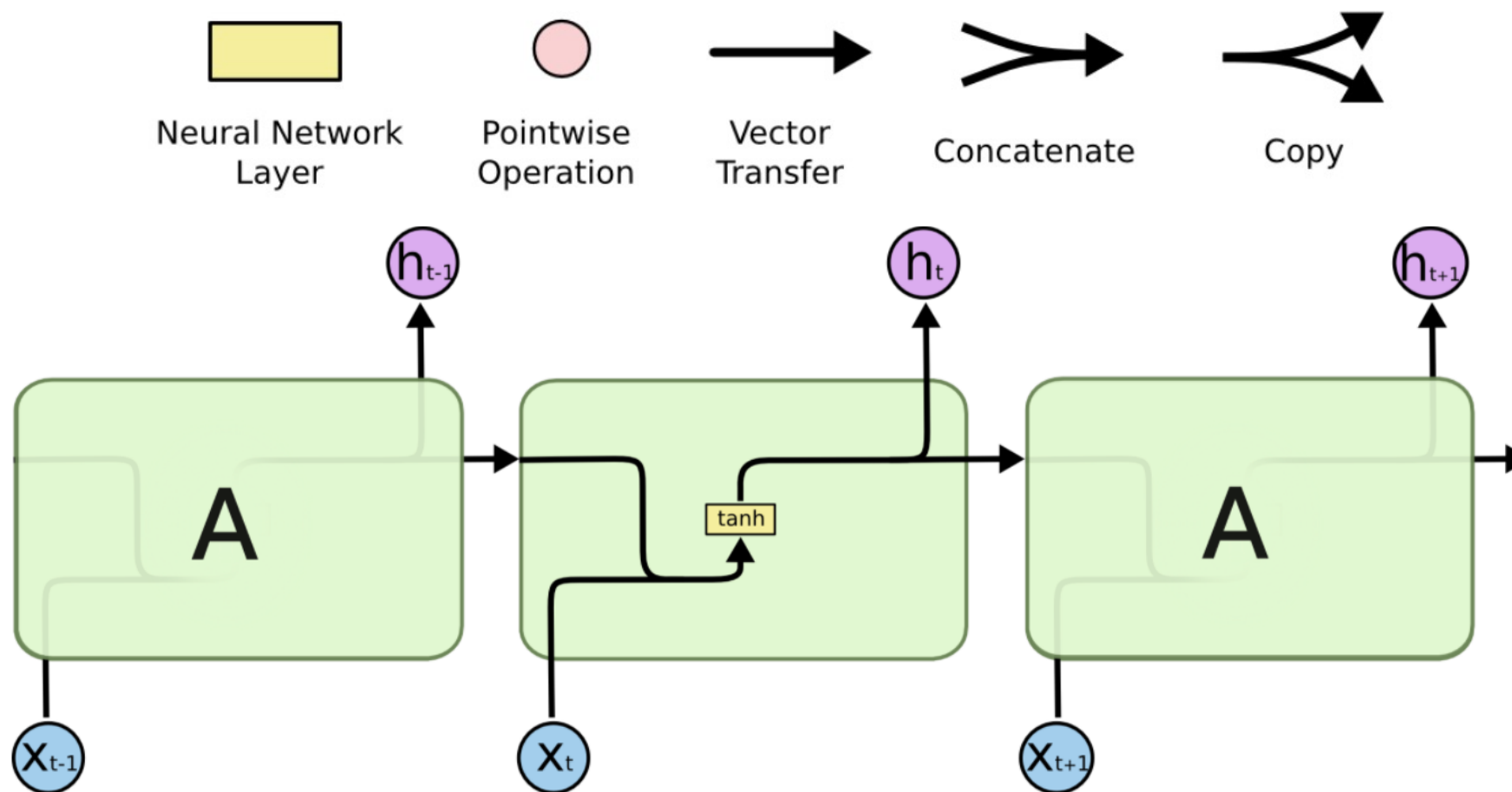
Jürgen Schmidhuber (2021, slightly updated 2023)
Pronounce: [You_again Shmidhoobuh](#)

AI Blog
Twitter: [@SchmidhuberAI](#)

The most cited neural networks all build on work done in my labs

Abstract. Modern Artificial Intelligence is dominated by artificial neural networks (NNs) and [deep learning](#).^[DL1-4] Foundations of the most popular NNs originated in my labs at TU Munich and IDSIA. Here I discuss: **(1) Long Short-Term Memory**^[LSTM0-17] (LSTM), the most cited NN of the 20th century, **(2) ResNet**, the most cited NN of the 21st century (which is an open-gated version of our earlier [Highway Net](#):^[HW1-3] the first working really deep feedforward NN), **(3) AlexNet and VGG Net**, the 2nd and 3rd most cited NNs of the 21st century (both building on our similar earlier [DanNet](#):^[GPUCNN1-9] the first deep convolutional NN^[CNN1-4] to win [image recognition competitions](#)), **(4) Generative Adversarial Networks**^[GAN0-1] (an instance of my earlier [Adversarial Artificial Curiosity](#)^{[AC90-20][DLH]}), and **(5) variants of Transformers** (Transformers with "linearized self-attention" are formally equivalent to my earlier [Fast Weight Programmers](#)).^[TR1-6]^{[FWP0-1,6][DLH]} Most of this started with our [Annus Mirabilis of 1990-1991](#)^[MIR] when compute was a million times more expensive than today. Back then we also laid foundations of Generative AI, publishing principles of **(4) GANs** (1990, now used for deepfakes),^{[AC90-20][DLH]} **(5) Transformers** (1991, the "T" in "ChatGPT" stands for "Transformer"),^{[TR1-6][FWP0-1,6][DLH]} and **(6) self-supervised pre-training** for deep NNs (1991, the "P" in "GPT" stands for "pre-trained").^{[UN][UN0-3]}

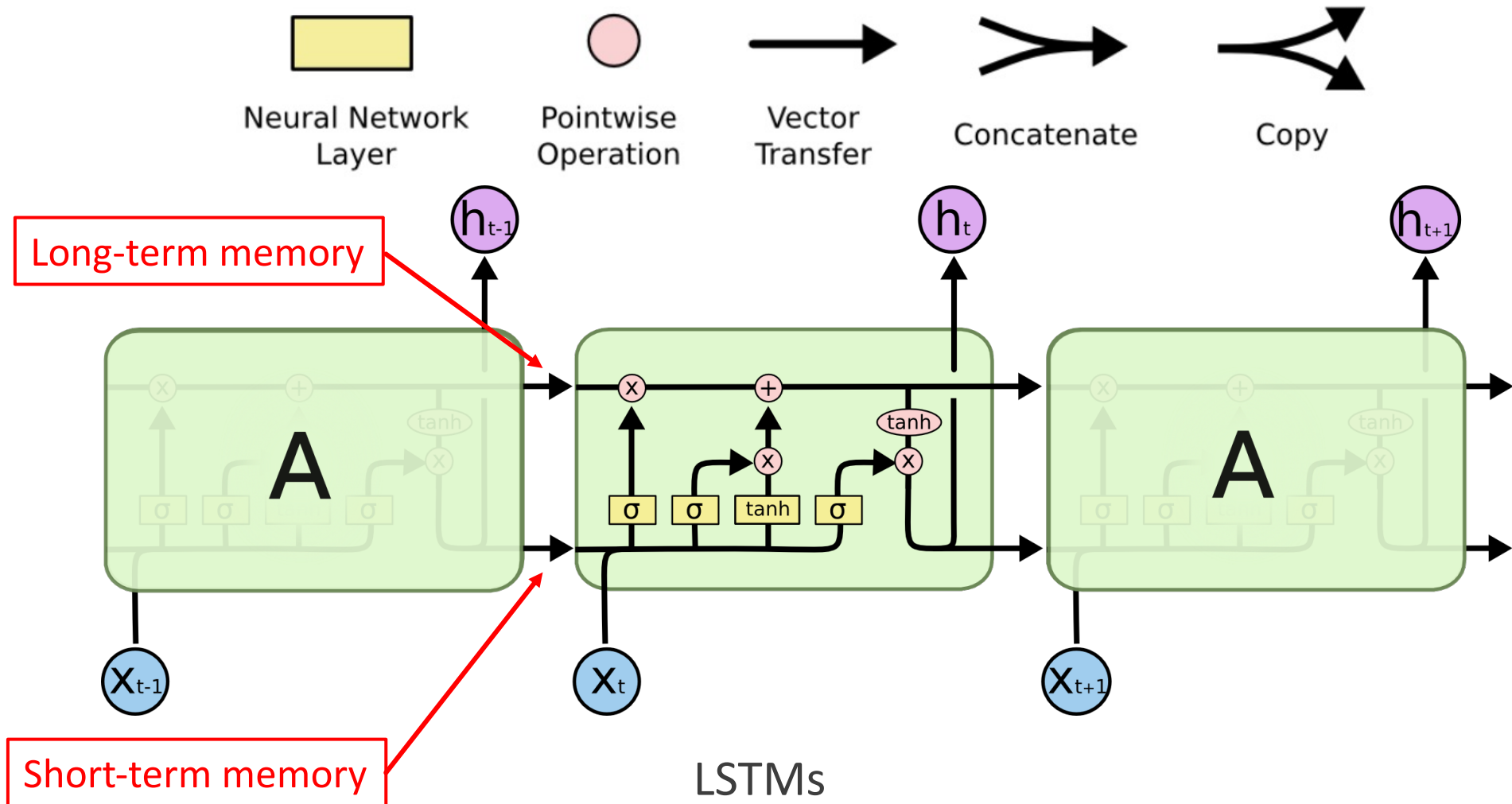
LSTMs



Standard form of vanilla RNNs



LSTMs

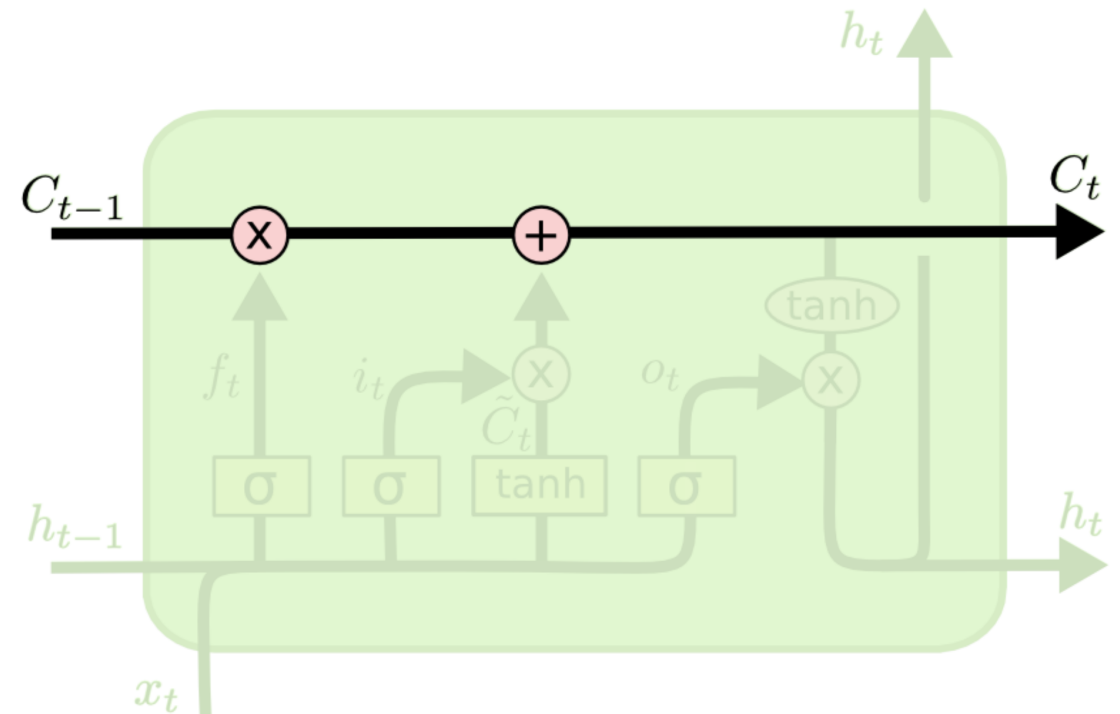


LSTMs



LSTMs: Cell State

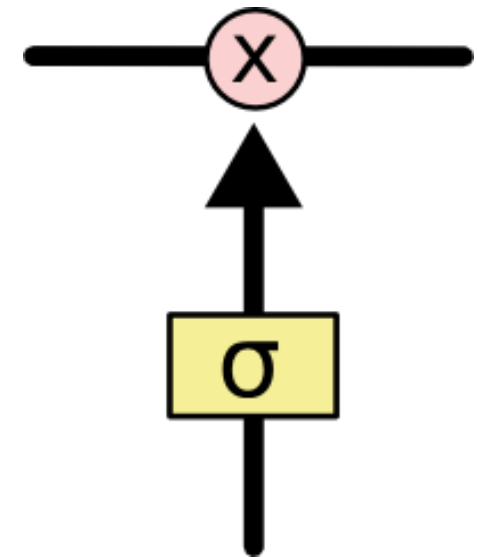
- The key to LSTMs is the **cell state**, the horizontal line running through the top of the diagram.
- The cell state is kind of like a conveyor belt (similar to the shortcut connection in ResNet).
- It runs straight down the entire chain, with only some **minor linear interactions (no concat)**.
- It's very easy for information to just flow along it unchanged.



LSTMs: Gates

- Gates are a way to **optionally** let information through.
 - Composed of a sigmoid neural net layer and a pointwise multiplication operation.
- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.
 - A value of zero means “let nothing through,” while a value of one means “let everything through!”
- An LSTM has three of these gates, to control and exploit the cell state.
 - Forget gate.
 - Input gate.
 - Output gate.

$$output = \sigma(W \cdot input + b)$$

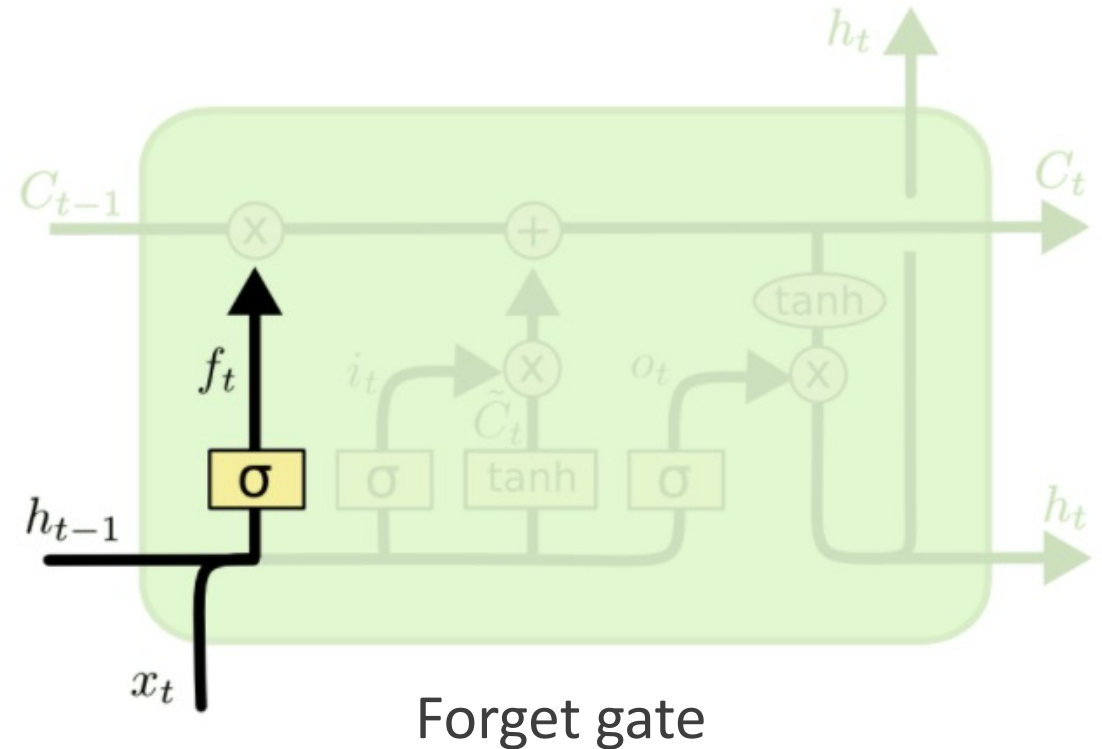


A gate in LSTMs



LSTMs: Forget Gate

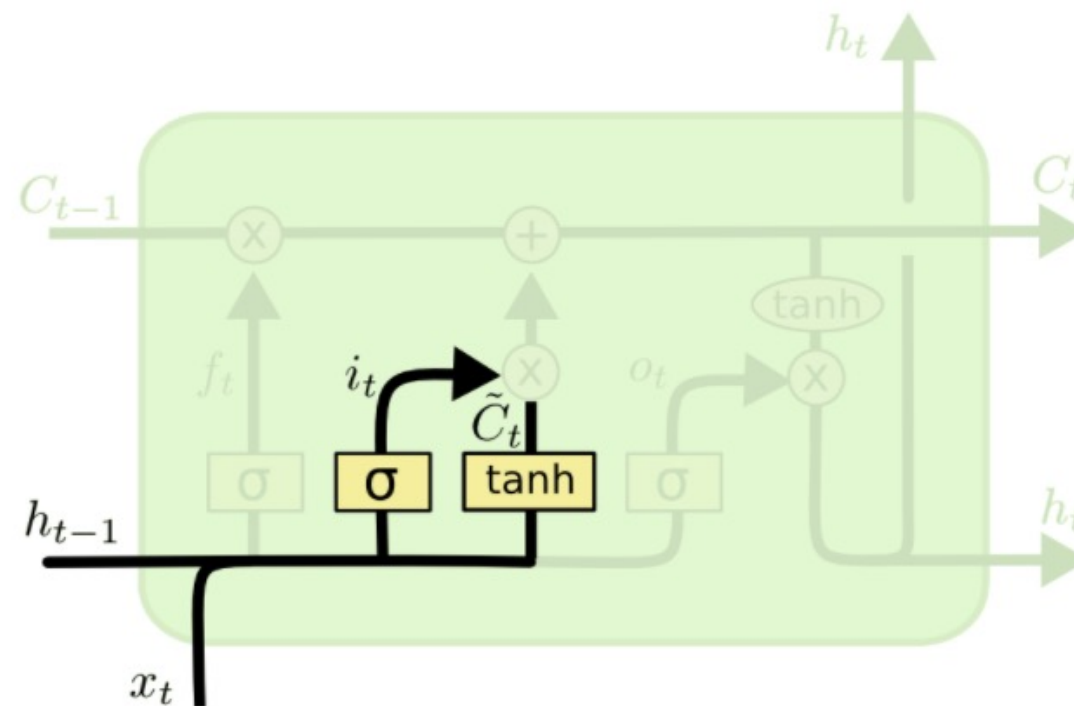
- Decide what information we're going to throw away from the cell state.
 - The output of sigmoid layer is between 0 and 1, and multiplied to each number in the cell state C_{t-1} .
- Example
 - The cell state might include the gender of the present subject, so that the correct pronouns can be used.
 - When we see a new subject, we want to forget the gender of the old subject.



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

LSTMs: Input Gate

- Decide what new information we're going to store in the cell state.
 - A sigmoid layer decides which values we'll update.
 - A tanh layer creates a vector of new candidate values.
- Example
 - Add the gender of the new subject to the cell state, to replace the old one we're forgetting.

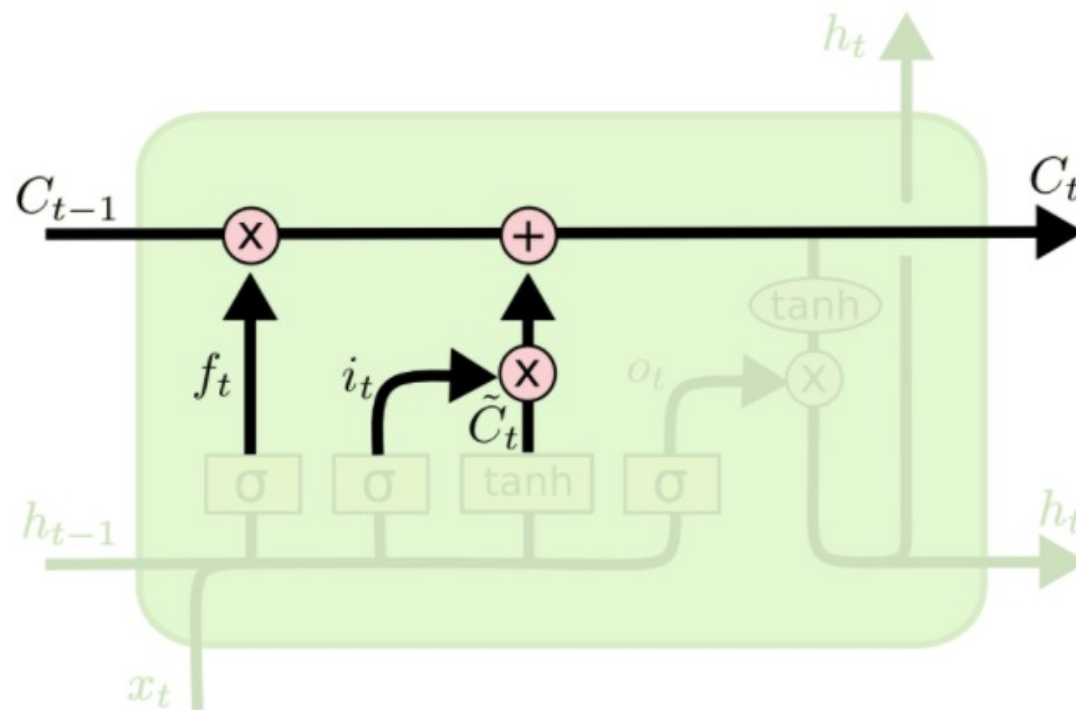


Input gate

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

LSTMs: Forget and Input Gate

- Update the old cell state, C_{t-1} , into the new cell state C_t .
 - Multiply the old state by f_t , forgetting the things we decided to forget earlier.
 - Add new information, which is transformed by \tilde{C}_t and selected by i_t .
- Example
 - Drop the information about the old subject's gender and add the new information.



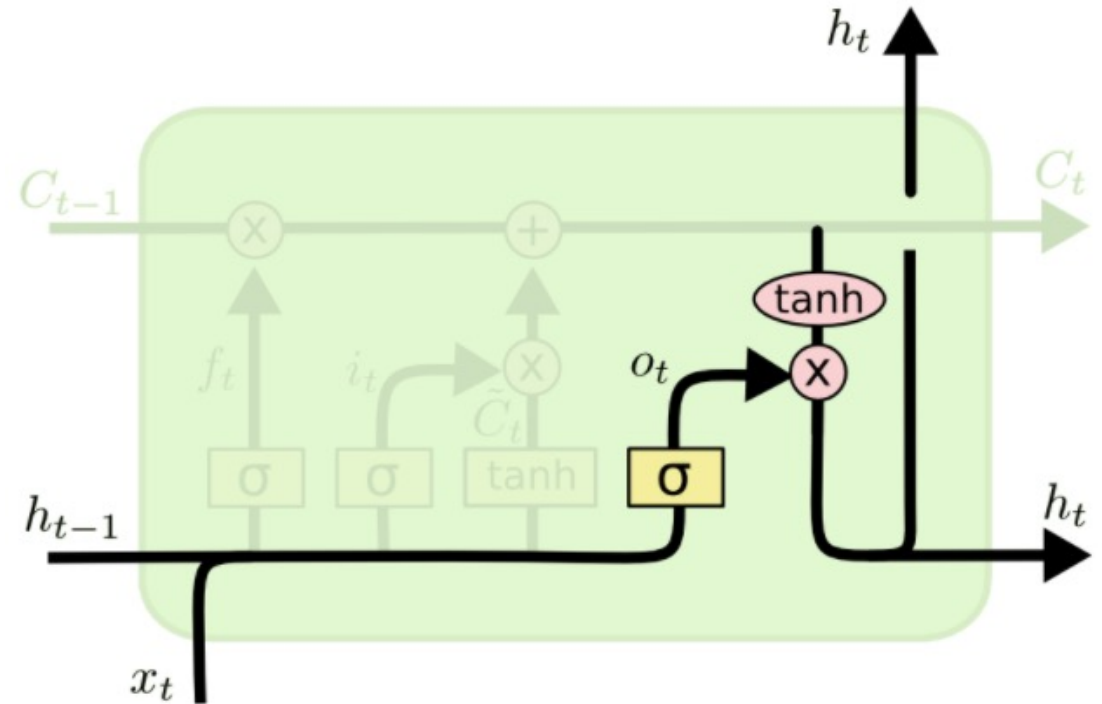
Update cell state

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



LSTMs: Output Gate

- Decide what we're going to output.
 - Run a sigmoid layer which decides what parts of the cell state we're going to output.
 - Put the cell state through tanh and multiply it by the output of the sigmoid layer.
- Example
 - It integrates the information of the new subject (e.g. singular or plural) with the scene or environment stored in the cell state to predict a coming relative verb.



Output gate

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

GRU

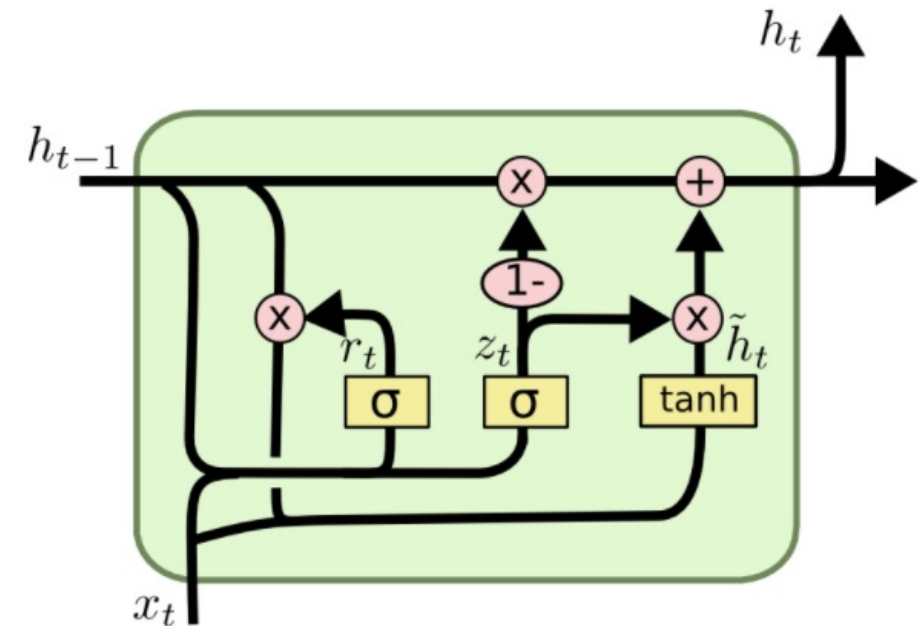
Learning phrase representations using RNN encoder-decoder for statistical machine translation

K Cho, B Van Merriënboer, C Gulcehre... - arXiv preprint arXiv ..., 2014 - arxiv.org

In this paper, we propose a novel neural network model called **RNN Encoder-Decoder** that consists of two **recurrent neural networks (RNN)**. One **RNN** encodes a sequence of symbols ...

☆ Save 77 Cite Cited by 26386 Related articles All 30 versions 88

- Gated Recurrent Unit (GRU) is a variant of LSTM.
- GRU performs similarly to LSTM but is computationally cheaper.
- Merges the cell state and hidden state.
- Combine the forget and input gates into a single update gate.



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

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

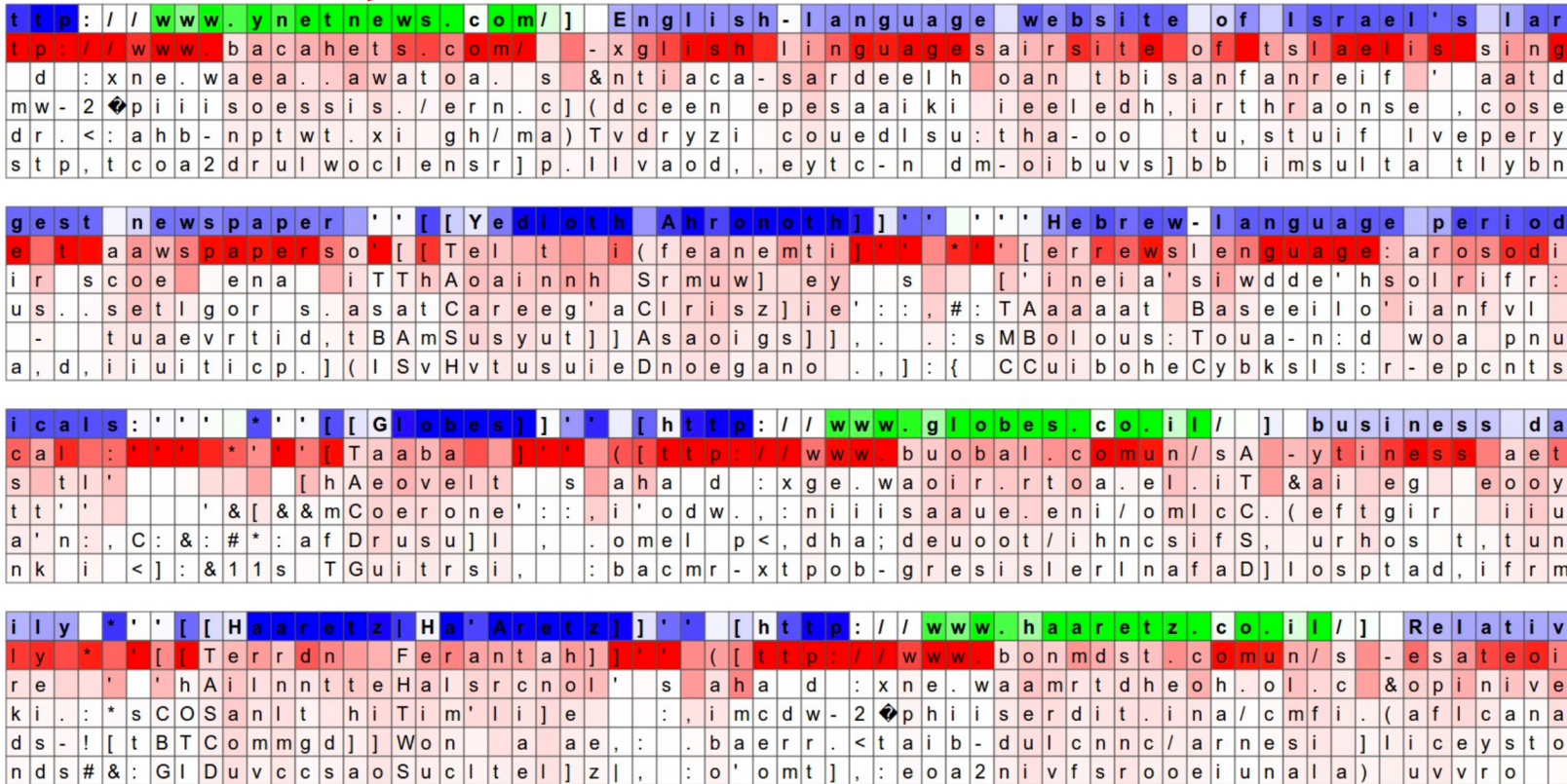
$$\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t])$$

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



Example

The LSTM is likely using this neuron to remember if it is inside a URL or not.

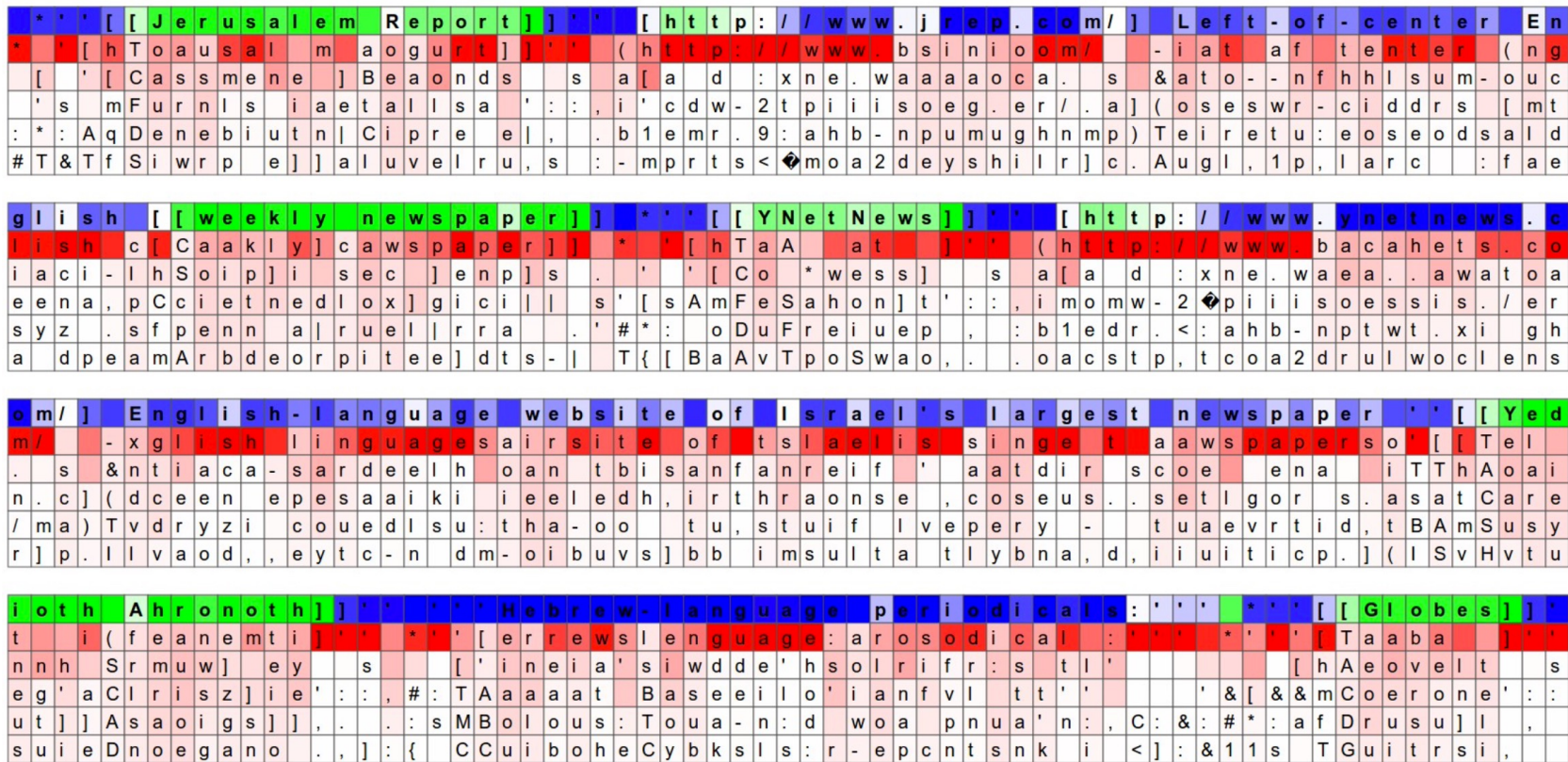


Next character prediction by LSTM. Red: top 5 prediction probability; Blue: negative value of some neuron in cell state; Green: positive value of some neuron in cell state.



Example

The highlighted neuron here gets very activated when the RNN is inside the [[]] markdown environment and turns off outside of it.

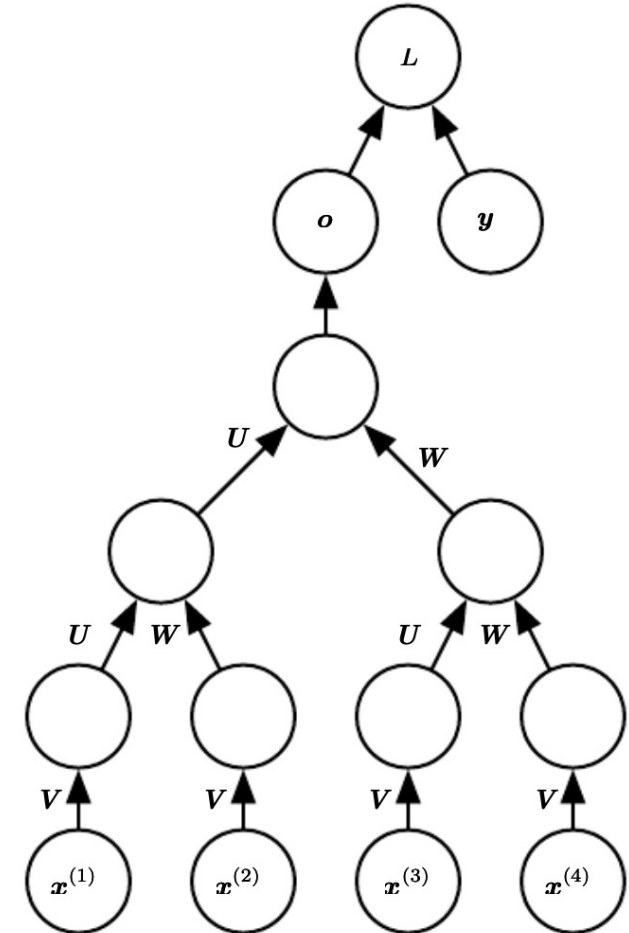


Next character prediction by LSTM. Red: top 5 prediction probability; Blue: negative value some neuron in cell state; Green: positive value of some neuron in cell state.



Recursive Neural Networks

- Recursive neural networks (not abbreviate as RNN to avoid confusion) is another generalization of recurrent networks.
- It is a **tree-like** structure, rather than the **chain-like** structure of RNNs.
- One advantage over RNNs is the association between any pair of inputs can be reduced from l to $O(\log l)$.
 - It might help deal with long-term dependencies.



Parameter sharing is also adopted



Case Study: Classifying Names with RNN by PyTorch

- Build and train a basic character-level RNN to classify names.
 - Input: words as a series of characters.
 - Output: scores of which language a name is from.

Case Study: Classifying Names with RNN by PyTorch

■ Training data:

jupyter Chinese.txt	
File	Edit View Language
1	Ang
2	Au-Yong
3	Bai
4	Ban
5	Bao
6	Bei
7	Bian
8	Bui
9	Cai
10	Cao
11	Cen
12	Chai
13	Chaim
14	Chan
15	Chang
16	Chao
17	Che
18	Chen
19	Cheng
20	Cheung

jupyter English.txt	
File	Edit View Language
1	Abbas
2	Abbey
3	Abbott
4	Abdi
5	Abel
6	Abraham
7	Abrahams
8	Abrams
9	Ackary
10	Ackroyd
11	Acton
12	Adair
13	Adam
14	Adams
15	Adamson
16	Adanet
17	Addams
18	Adderley
19	Addinall
20	Addis

jupyter Japanese.txt	
File	Edit View Language
1	Abe
2	Abukara
3	Adachi
4	Aida
5	Aihara
6	Aizawa
7	Ajibana
8	Akaike
9	Akamatsu
10	Akatsuka
11	Akechi
12	Akera
13	Akimoto
14	Akita
15	Akiyama
16	Akutagawa
17	Amagawa
18	Amaya
19	Amori
20	Anami

jupyter Russian.txt	
File	Edit View Language
1	Ababko
2	Abaev
3	Abagyan
4	Abaidulin
5	Abaidullin
6	Abaimoff
7	Abaimov
8	Abakeliya
9	Abakovsky
10	Abakshin
11	Abakumoff
12	Abakumov
13	Abakumtsev
14	Abakushin
15	Abalakin
16	Abalakoff
17	Abalakov
18	Abaleshev
19	Abalihin
20	Abalikhin



- Turning names into Tensors.
- Use one-hot vector of size $\langle 1 \times n_letters \rangle$.
 - e.g. "b" = [0, 1, 0, 0, 0, ...].
- To make a word we join a bunch of those into a 2D matrix $\langle line_length \times 1 \times n_letters \rangle$.
 - The dimension 1 in the middle is the batch size.

```

all_letters = string.ascii_letters + " .,;"
n_letters = len(all_letters)

# Find letter index from all_letters, e.g. "a" = 0
def letterToIndex(letter):
    return all_letters.find(letter)

# Just for demonstration, turn a letter
# into a <1 x n_letters> Tensor
def letterToTensor(letter):
    tensor = torch.zeros(1, n_letters)
    tensor[0][letterToIndex(letter)] = 1
    return tensor

# Turn a line into a <line_length x 1 x n_letters>,
# or an array of one-hot letter vectors
def lineToTensor(line):
    tensor = torch.zeros(len(line), 1, n_letters)
    for li, letter in enumerate(line):
        tensor[li][0][letterToIndex(letter)] = 1
    return tensor

```

```

print(letterToTensor('J'))
print(lineToTensor('Jones').size())

tensor([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0.]])
torch.Size([5, 1, 57])

```


■ Creating the network

```
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.tanh = nn.Tanh()
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        hidden = self.tanh(hidden)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```

```
input = lineToTensor('Albert')
hidden = torch.zeros(1, n_hidden)

output, next_hidden = rnn(input[0], hidden)
print(output)
print(next_hidden)
```

```
tensor([[ -2.8816, -2.8425, -2.8608, -2.9792, -2.8023, -2.8851, -2.8717, -2.9368,
        -2.9888, -2.8223, -2.9434, -2.9038, -2.9192, -2.7998, -2.9649, -2.8405,
        -2.9298, -2.8828]], grad_fn=<LogSoftmaxBackward>)
tensor([[ 6.1259e-02, -4.9072e-02,  1.7027e-02,  6.4582e-02, -5.1130e-02,
        -6.0277e-02,  5.9739e-02,  1.6188e-02, -1.2441e-01,  1.9956e-02,
        -8.2753e-02,  6.4828e-02,  1.4753e-02,  5.0268e-02, -1.8191e-02,
         4.7226e-02,  1.8563e-02, -1.9493e-02,  1.1526e-01,  1.0337e-01,
        -2.1714e-02,  1.0513e-01,  3.8046e-03, -5.6472e-02,  4.3164e-02,
        -3.5685e-02,  5.2062e-02, -1.3111e-01, -3.6113e-02, -1.6371e-02,
         1.1246e-01,  4.7487e-03,  2.0831e-02, -9.0459e-02, -5.6609e-03,
        -5.5420e-02,  1.8348e-02,  5.0092e-02, -1.0727e-01,  2.7786e-02,
        -6.2168e-03, -8.4481e-03, -1.1548e-02, -3.0568e-02,  1.9215e-02,
         2.3361e-02,  3.2877e-02,  1.0187e-01, -6.9983e-03, -4.7263e-02,
         9.2099e-02, -1.1344e-02, -1.0038e-02, -7.0406e-02, -3.3504e-02,
         3.6116e-02, -4.6870e-03, -9.5091e-03,  3.8454e-02, -5.6868e-02,
         8.5306e-03,  8.5978e-02, -2.5189e-02,  2.3690e-02,  1.2213e-01,
         1.9925e-02,  4.9675e-02, -7.5523e-05, -2.8934e-02,  1.8377e-02,
         4.6675e-02, -1.3534e-02,  9.0922e-02,  5.3885e-02,  6.8588e-02,
         2.7824e-02,  9.8373e-02,  5.5584e-02, -7.0373e-03, -2.2631e-03,
         1.3586e-01,  7.4371e-02, -1.3851e-02,  5.3747e-02, -4.0922e-02,
         5.3521e-02, -9.3251e-03, -5.7613e-02,  3.9844e-02,  7.5398e-02,
         5.4752e-02, -4.3124e-02,  3.2717e-02, -4.8338e-03, -8.6761e-02,
         9.8632e-02, -7.4960e-02, -3.4206e-02, -1.8842e-02, -5.0261e-02,
         7.6002e-02, -4.6946e-03, -9.0249e-02, -2.9031e-04, -1.1506e-01,
        -1.2587e-02,  5.4591e-02,  7.3432e-02, -6.4257e-02, -1.6510e-03,
        -5.2874e-03,  4.1667e-02, -8.3270e-02, -1.7023e-02,  5.4377e-02,
        -3.9851e-02, -8.3871e-02,  2.3032e-02, -7.6733e-02,  7.1338e-02,
         1.2173e-01, -9.5176e-02, -8.9871e-02, -5.5411e-02, -1.9085e-02,
        -1.8339e-02,  9.5004e-02, -5.4389e-02]], grad_fn=<TanhBackward>)
```

Output is <1 x n_categories>.
Every item is the likelihood of that category.

Hidden state

```
learning_rate = 0.005
# If you set this too high, it might explode.
# If too low, it might not learn

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(p.grad.data, alpha=-learning_rate)

    return output, loss.item()
```

Only the last output is returned.


```


for iter in range(1, n_iters + 1):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    output, loss = train(category_tensor, line_tensor)
    current_loss += loss

    # Print iter number, loss, name and guess
    if iter % print_every == 0:
        guess, guess_i = categoryFromOutput(output)
        correct = '✓' if guess == category else 'X (%s)' % category
        print('%d %d%% (%s) %.4f %s / %s %s' % (iter, iter / n_iters * 100, \
            timeSince(start), loss, line, guess, correct))

    # Add current loss avg to list of losses
    if iter % plot_every == 0:
        all_losses.append(current_loss / plot_every)
        current_loss = 0

```

Feed a name (a line)
into RNN each time



```

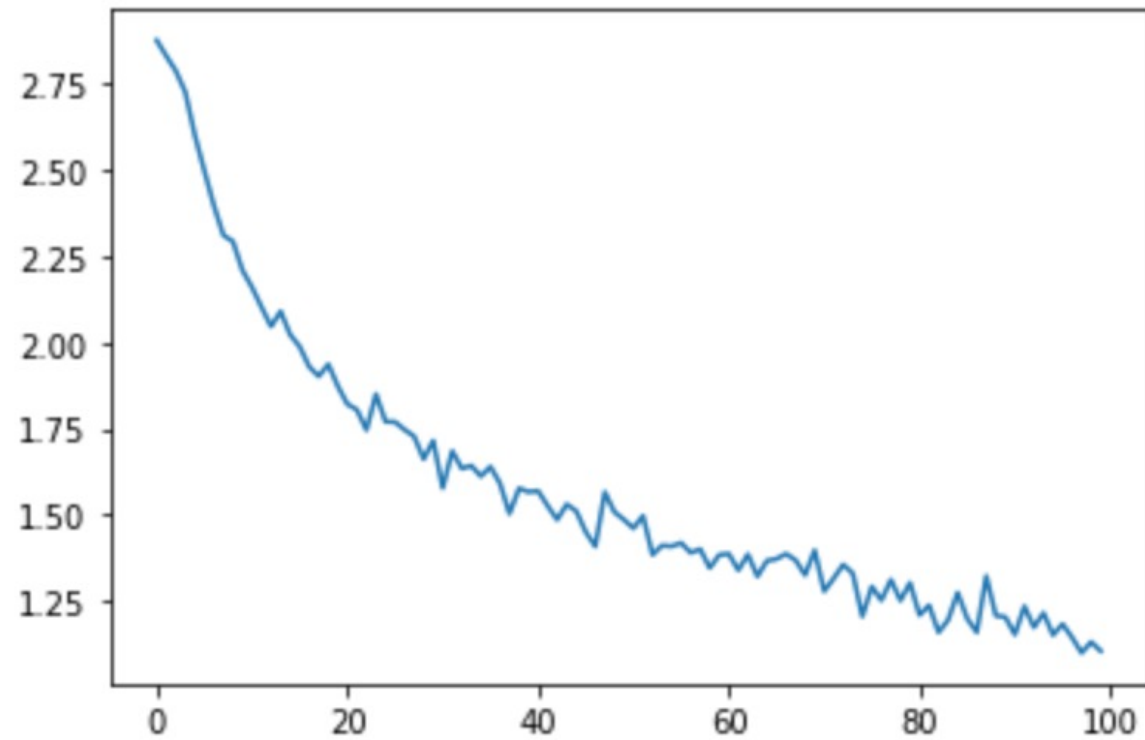
5000 5% (0m 11s) 2.6765 Nunes / Greek X (Portuguese)
10000 10% (0m 21s) 2.2783 Segher / German X (Dutch)
15000 15% (0m 32s) 1.6745 Mcmillan / Irish X (Scottish)
20000 20% (0m 42s) 0.1753 Hashimoto / Japanese ✓
25000 25% (0m 53s) 0.8476 Sciacchitano / Italian ✓
30000 30% (1m 4s) 0.5257 Dubhain / Irish ✓
35000 35% (1m 15s) 1.8122 Mustafa / Japanese X (Arabic)
40000 40% (1m 26s) 1.9135 Kwei / Korean X (Chinese)
45000 45% (1m 37s) 0.1803 Fukunaka / Japanese ✓
50000 50% (1m 48s) 0.8750 Xiong / Chinese ✓
55000 55% (1m 58s) 1.3530 Lebeau / French ✓
60000 60% (2m 9s) 0.4622 Zhan / Chinese ✓
65000 65% (2m 20s) 4.1396 Re / Korean X (Italian)
70000 70% (2m 30s) 2.2434 Kennedy / English X (Irish)
75000 75% (2m 41s) 0.5489 Fleming / Scottish ✓
80000 80% (2m 52s) 0.5621 Macdonald / Scottish ✓
85000 85% (3m 4s) 2.1335 Rios / English X (Portuguese)
90000 90% (3m 15s) 1.1279 Larue / French ✓
95000 95% (3m 26s) 3.4025 Hakimi / Japanese X (Arabic)
100000 100% (3m 37s) 0.2672 Calogerakis / Greek ✓

```

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
```

```
plt.figure()
plt.plot(all_losses)
```

```
[<matplotlib.lines.Line2D at 0x7fa929c0b470>]
```



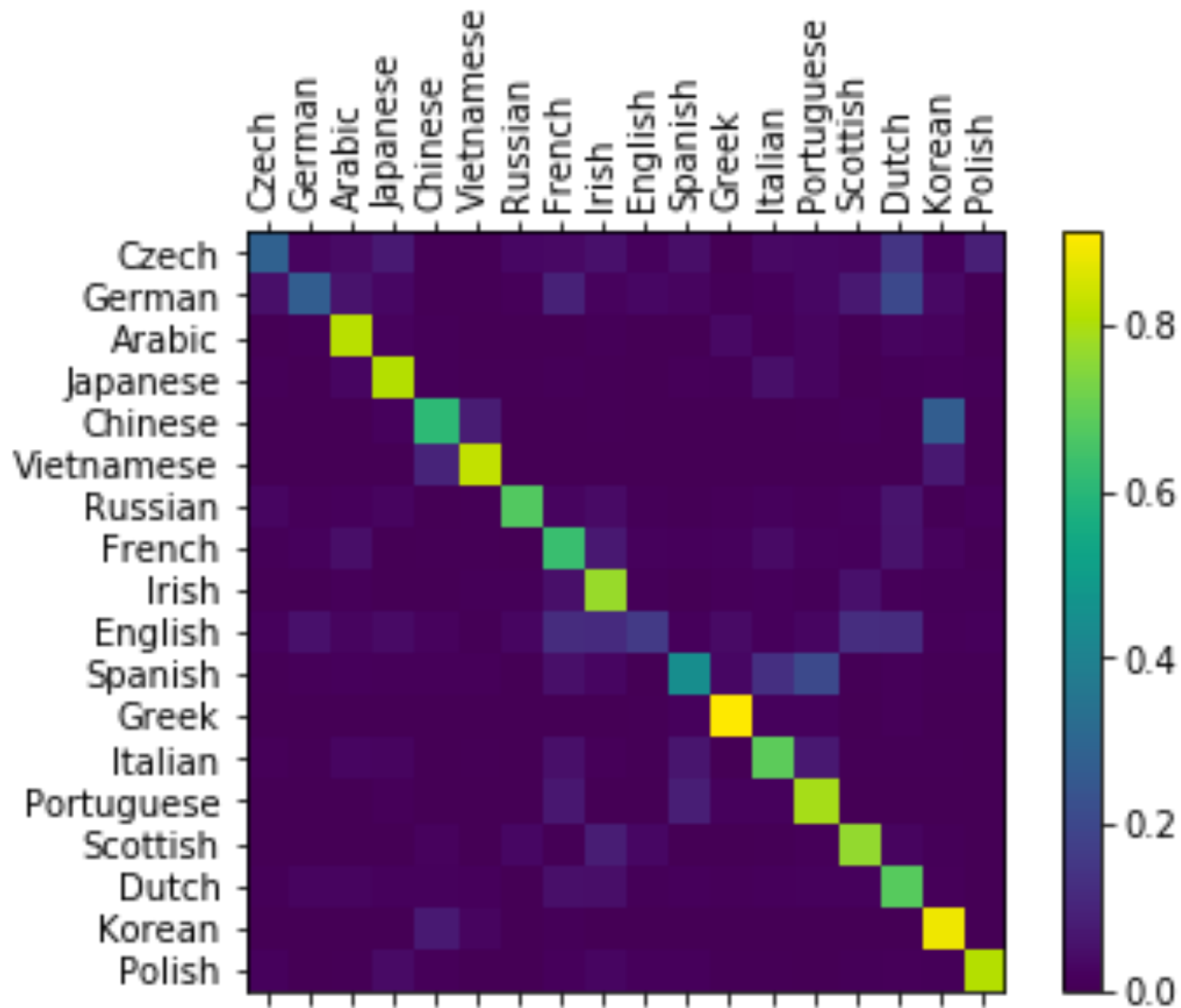
- Create a confusion matrix, indicating for every actual language (rows) which language the network guesses (columns).

```
# Keep track of correct guesses in a confusion matrix
confusion = torch.zeros(n_categories, n_categories)
n_confusion = 10000

# Just return an output given a line
def evaluate(line_tensor):
    hidden = rnn.initHidden()
    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)
    return output

# Go through a bunch of examples and record which are correctly guessed
for i in range(n_confusion):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    output = evaluate(line_tensor)
    guess, guess_i = categoryFromOutput(output)
    category_i = all_categories.index(category)
    confusion[category_i][guess_i] += 1

# Normalize by dividing every row by its sum
for i in range(n_categories):
    confusion[i] = confusion[i] / confusion[i].sum()
```



```

def predict(input_line, n_predictions=3):
    print('\n> %s' % input_line)
    with torch.no_grad():
        output = evaluate(lineToTensor(input_line))

        # Get top N categories
        topv, topi = output.topk(n_predictions, 1, True)
        predictions = []

        for i in range(n_predictions):
            value = topv[0][i].item()
            category_index = topi[0][i].item()
            print('({:.2f}) %s' % (value, all_categories[category_index]))
            predictions.append([value, all_categories[category_index]])

predict('Jackson')
predict('Satoshi')
predict('Lu')
predict('Yang')

```

```

> Jackson
(-0.09) Scottish
(-3.81) Greek
(-3.96) English

> Satoshi
(-0.56) Arabic
(-1.14) Japanese
(-3.06) Polish

> Lu
(-0.12) Vietnamese
(-2.59) Korean
(-4.23) Chinese

> Yang
(-0.13) Korean
(-2.17) Chinese
(-5.87) Vietnamese

```

???

Conclusion

After this lecture, you should know:

- What is the basic structure of RNNs.
- What is the hidden state.
- How do RNNs handle sequential data.
- What is attention and how does it help?
- What is the problem of long-term dependencies?
- What is the basic idea of LSTM?

Suggested Reading

- Deep learning textbook chapter 9
- The Unreasonable Effectiveness of Recurrent Neural Networks
- Understanding LSTM Networks
- Visualizing A Neural Machine Translation Model
- Show, Attend and Tell: Neural Image Caption Generation with Visual Attention
- Neural Machine Translation By Jointly Learning To Align And Translate

Thank you!

- Any question?
- Don't hesitate to send email to me for asking questions and discussion. 😊