# Recurrent neural network (RNN), Long short-term memory (LSTM), Gated recurrent unit (GRU)

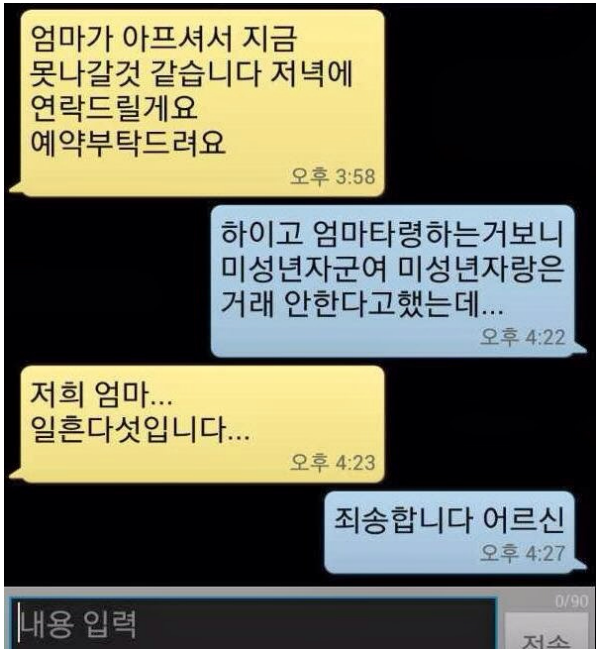
서울대학교병원 정보화실

고태훈 (taehoonko@snuh.org)

최세원 (swc@snuh.org)

# Contents

# Sequence data?




# Sequence data!

- **배열의 순서에 따라 관측된 현상을 표현한 데이터**

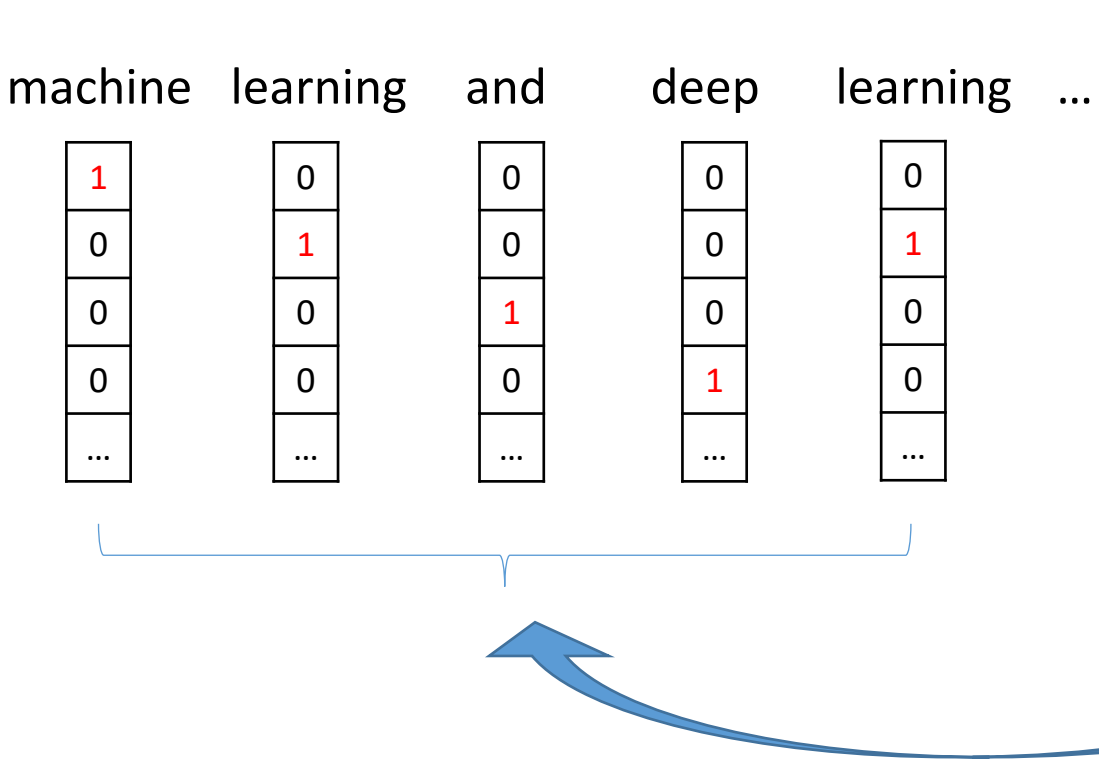
$$(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)} \ldots, \boldsymbol{x}^{(T)})$$

where each data point $\boldsymbol{x}^{(t)}$, which is observed at *time step* $t$, is a real value or real-valued vector/tensor.

- The sequences have an explicit or implicit temporal aspect.
- The sequences may be of finite or countably infinite length.

# Sentence to sequence form

- **How to represent "*Machine learning and deep learning are interesting.*"**
  - Word-based (one-hot vector)

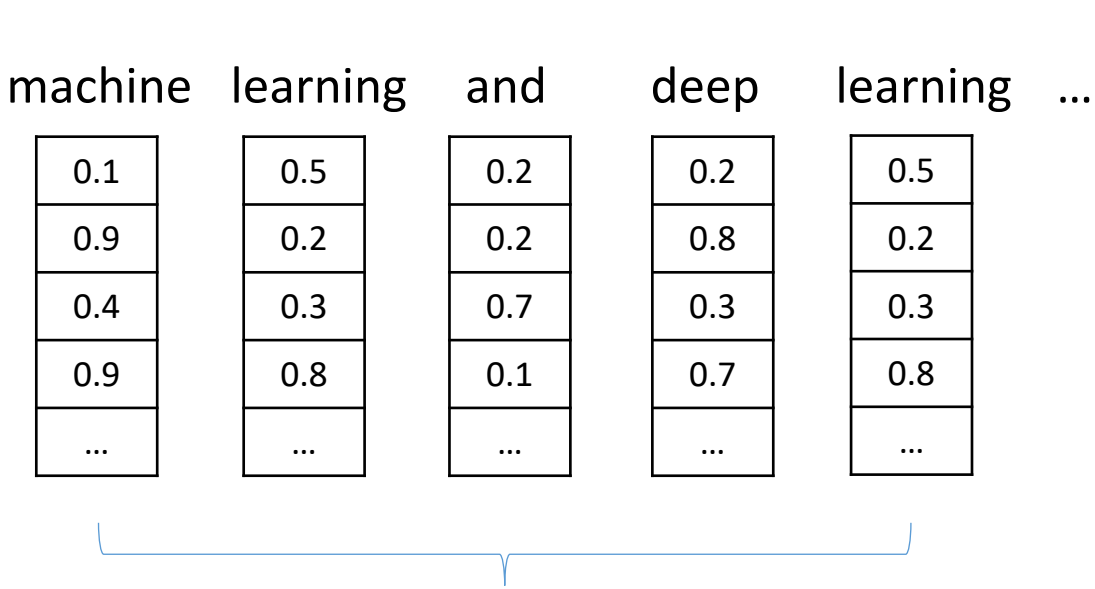



| word | index |
|---|---|
| machine | 0 |
| learning | 1 |
| and | 2 |
| deep | 3 |
| are | 4 |
| interesting | 5 |
| … | … |

convert each index to an one-hot vector

# Sentence to sequence form

- **How to represent "*Machine learning and deep learning are interesting.*"**
  - Word-based (embedded dense vector)

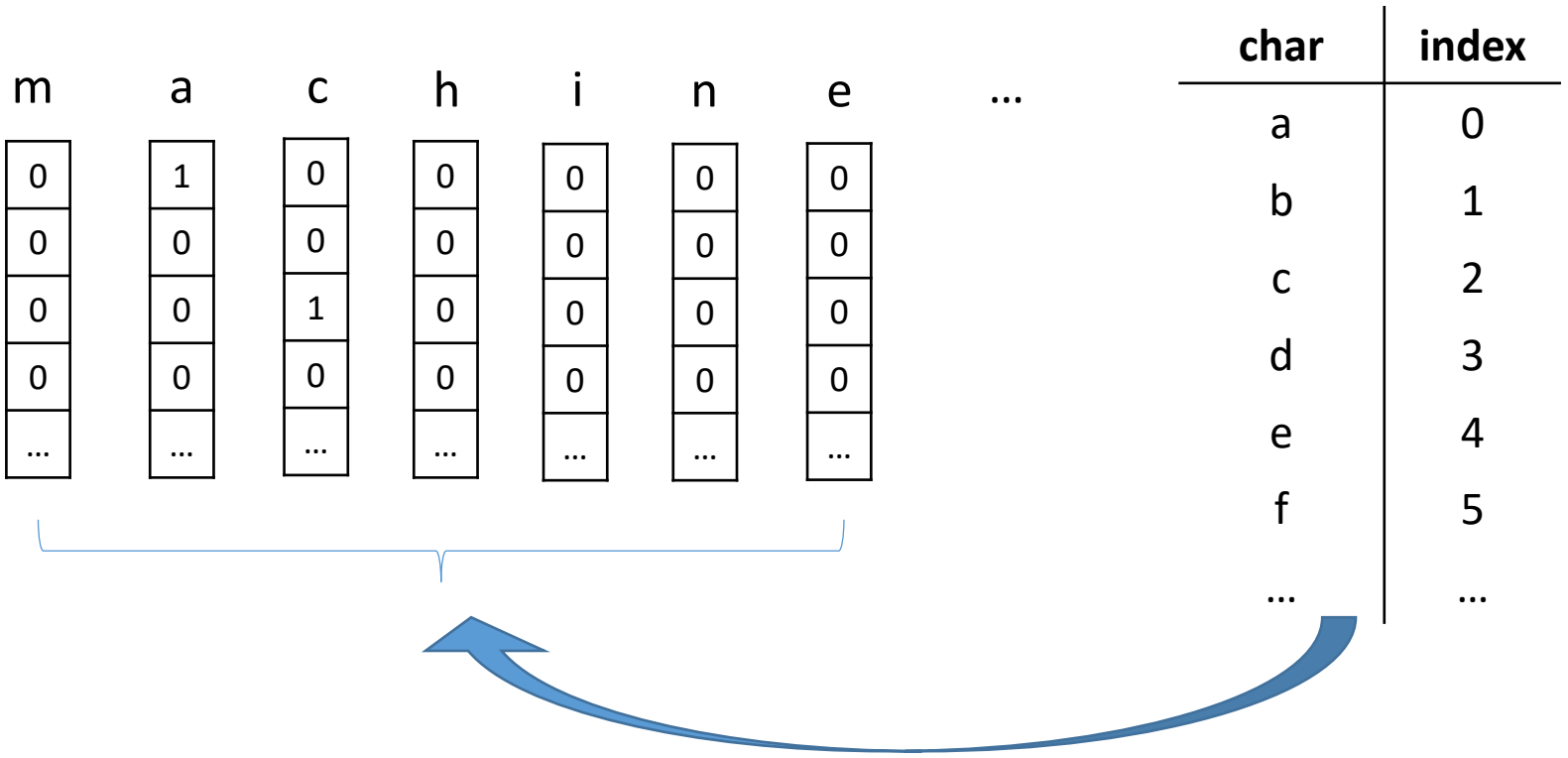



| word | v1 | v2 | … |
|---|---|---|---|
| machine | 0.1 | 0.9 | … |
| learning | 0.5 | 0.2 | … |
| and | 0.2 | 0.2 | … |
| deep | 0.2 | 0.8 | … |
| are | 0.9 | 0.7 | … |
| interesting | 0.5 | 0.4 | … |
| … | … | … | … |

lookup an embedded vector table

# Sentence to sequence form

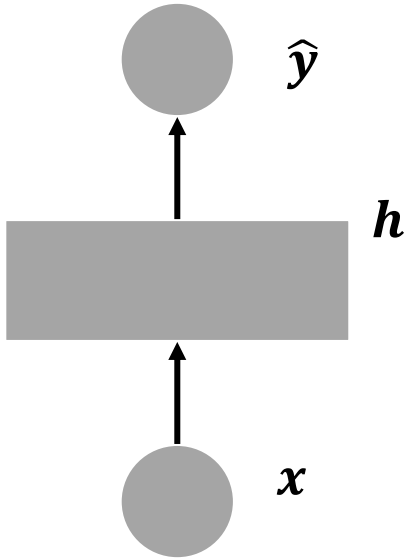
- **How to represent *"Machine learning and deep learning are interesting."***
  - Character-based (one-hot vector)



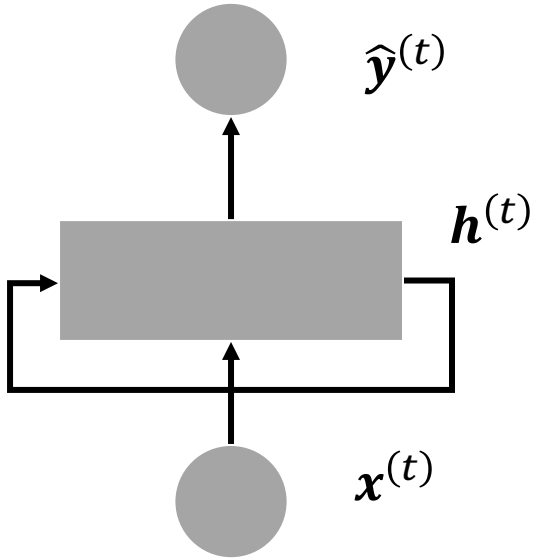
# Contents

# Recurrent neural network (RNN)

- **Recurrent neural networks**
  - feedforward neural networks augmented by the inclusion of edges that span adjacent time steps, introducing a notion of time to the model (Lipton et al., 2015).



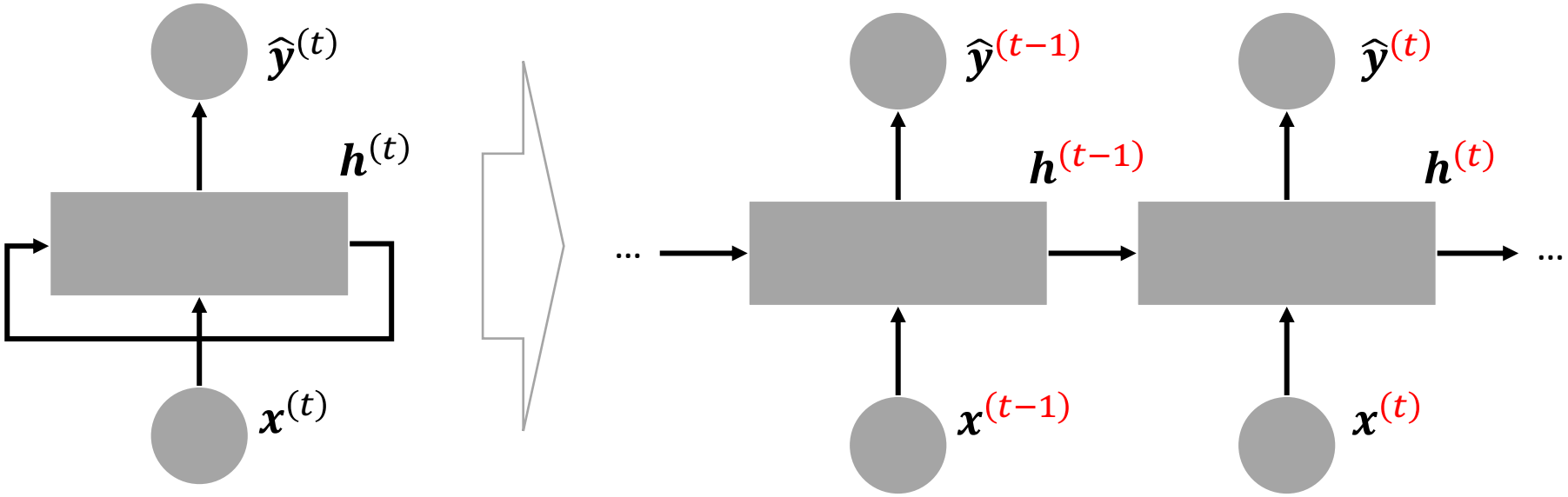

<Feedforward network>




<Recurrent neural network>

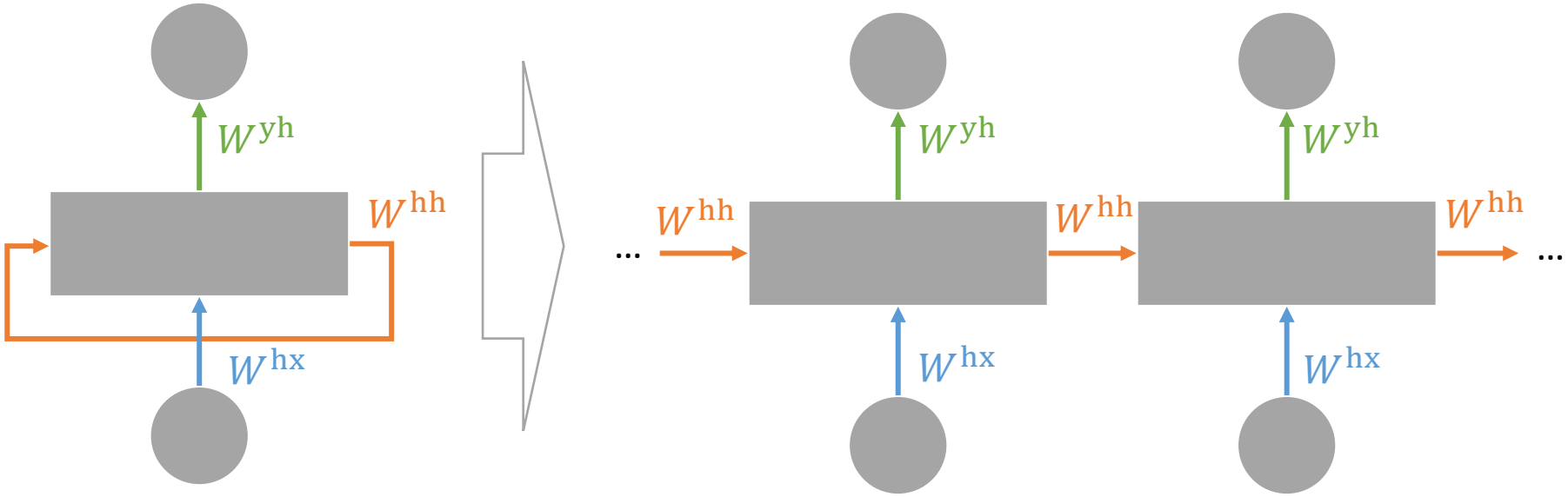
# Recurrent neural network (RNN)

- The data points, hidden node values and outputs: Time-variant



# Recurrent neural network (RNN)

- **Weights and biases: Time-invariant.**



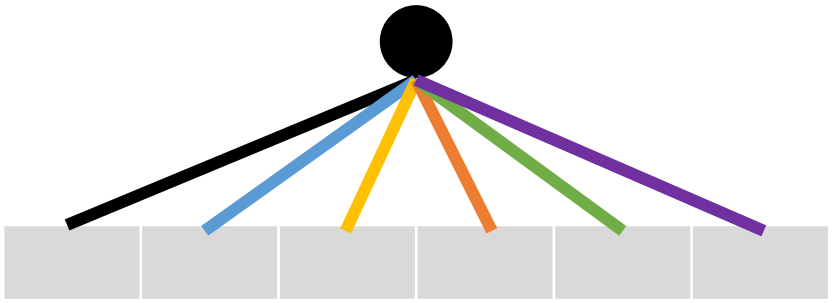
# (참고) Parameter sharing

- **Parameter sharing**
  - One of the early ideas found in machine learning
  - e.g., Hidden Markov model (HMM)

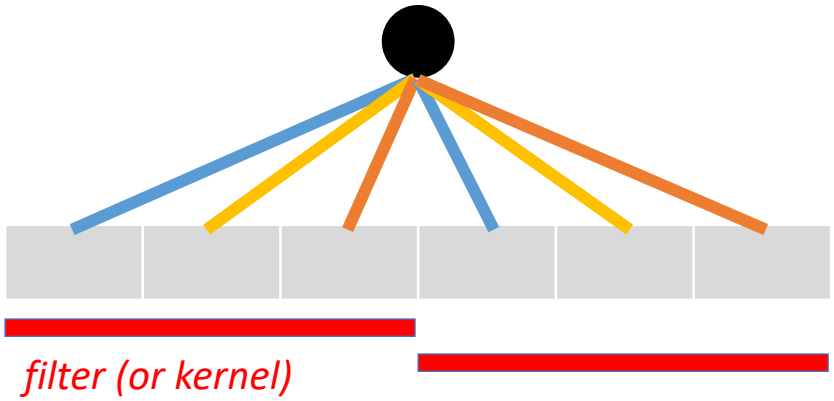
    ➔ The state transition matrix $P(s(t)|s(t-1))$ is reused for every time step $t$.
  - It allows to extend and apply the model to examples of different forms and generalize cross them.

# (참고) Weight sharing in CNNs

- **Benefits of using CNNs**
  - They use fewer parameters (weights) to learn than a fully connected network.



Number of weights: 6

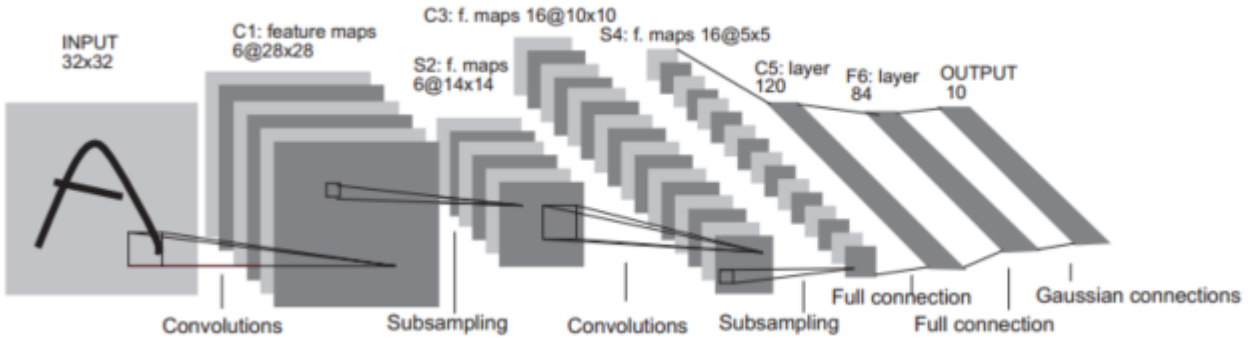



Number of weights: 3
(***weights sharing***)

→ **1x3** filter with stride **3**

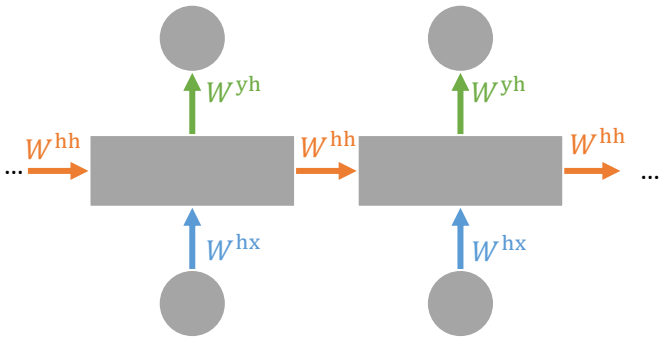
# (참고) Weight sharing

- **CNN**
  - Weight sharing gives robustness and efficiency.



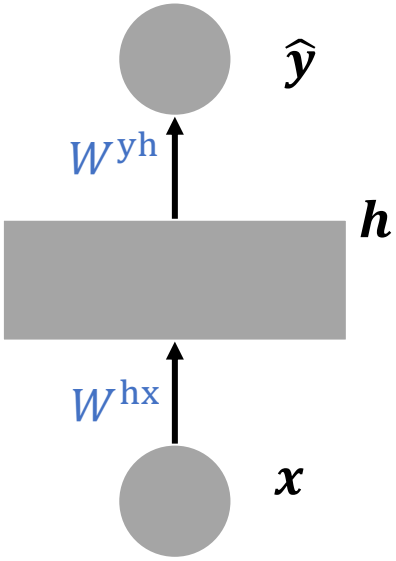

source: LeNet-5

- **RNN**
  - The same weights are used at different time steps
  - This allows much better generalization properties.

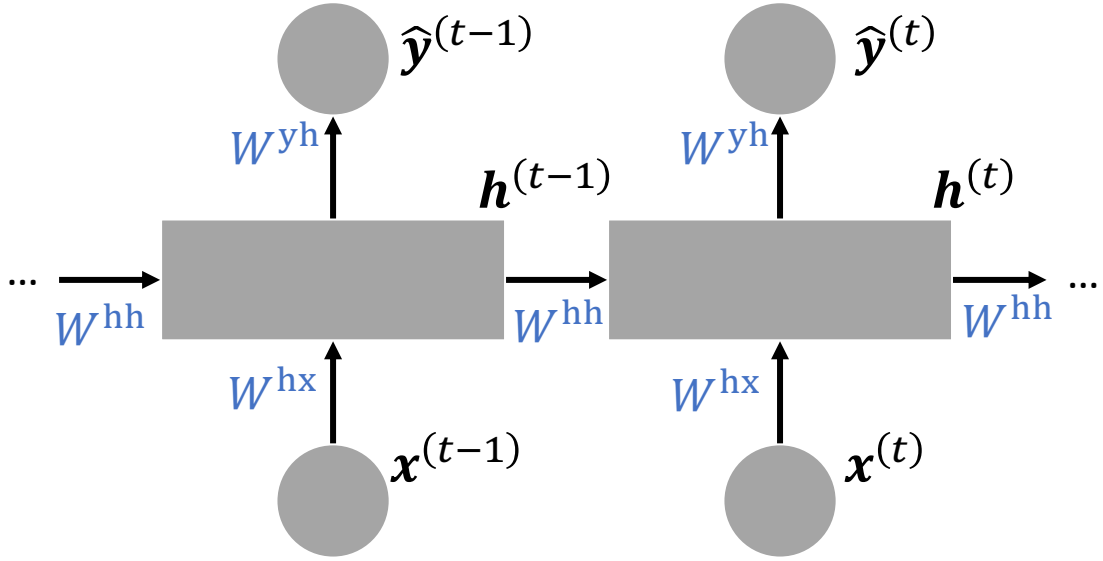


# Recurrent neural network (RNN)

- **Formulation**




<Feedforward network>




<Recurrent neural network>

$$\boldsymbol{h} = f(W^{\mathrm{hx}}\boldsymbol{x} + b^{\mathrm{h}})$$

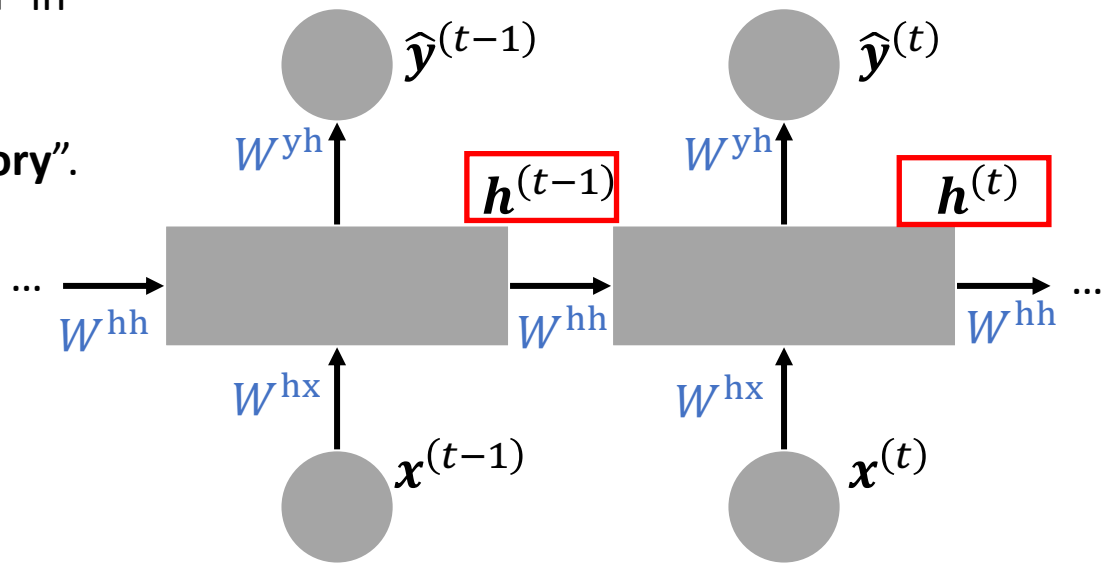
$$\hat{\boldsymbol{y}} = g(W^{\mathrm{yh}}\boldsymbol{h} + b^{\mathrm{y}})$$

$$\boldsymbol{h}^{(t)} = f(W^{\mathrm{hx}}\boldsymbol{x}^{(t)} + W^{\mathrm{hh}}\boldsymbol{h}^{(t-1)} + b^{\mathrm{h}})$$

$$\hat{\boldsymbol{y}}^{(t)} = g(W^{\mathrm{yh}}\boldsymbol{h}^{(t)} + b^{\mathrm{y}})$$

# Recurrent neural network (RNN)

- **State (vector):** $\boldsymbol{h}^{(t)}$
  - RNNs maintain a '**state vector**' in their each hidden layer.
  - It is considered as the "**memory**".




<Recurrent neural network>

The states (are expected to) contain information about the history of all the past elements of the sequence.

RNNs are better for tasks that involve sequential inputs such as speech, language, and genome.

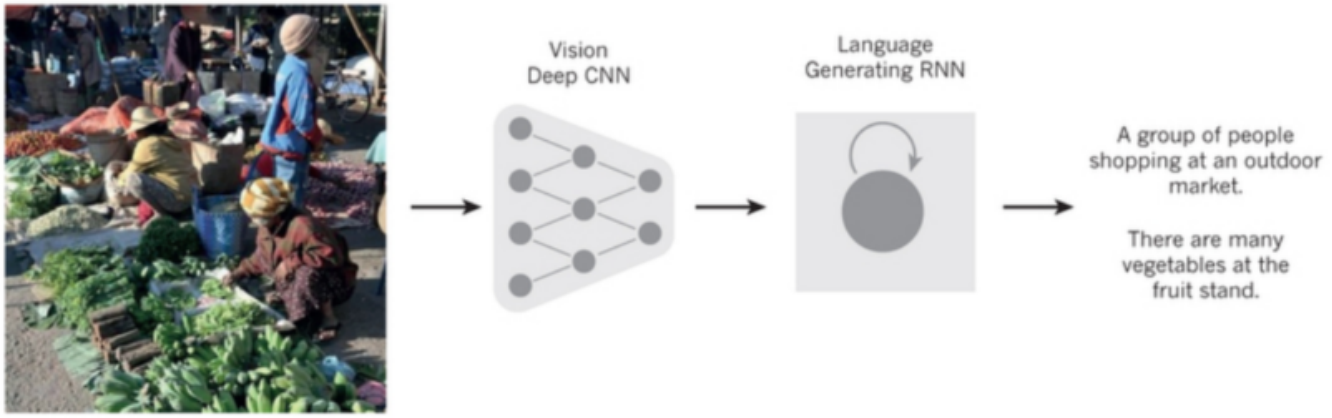
$$\boldsymbol{h}^{(t)} = f(W^{\mathrm{hx}}\boldsymbol{x}^{(t)} + W^{\mathrm{hh}}\boldsymbol{h}^{(t-1)} + b^{\mathrm{h}})$$

$$\hat{\boldsymbol{y}}^{(t)} = g(W^{\mathrm{yh}}\boldsymbol{h}^{(t)} + b^{\mathrm{y}})$$

# RNN: Applications

- **Natural language processing**
  - language modeling
  - machine translation
  - syntactic parsing of sentences
- **Speech/acoustic/music**
  - speech recognition
  - acoustic modeling
  - polyphonic music modeling
- **Bioinformatics**
  - regulatory genomics
  - protein structure prediction
- **Computer vision**
  - handwriting recognition
  - scene understanding (annotation/segmentation/classification)
  - automated image caption generation

# Example: Image caption generation




A woman is throwing a **frisbee** in a park.

A **dog** is standing on a hardwood floor.

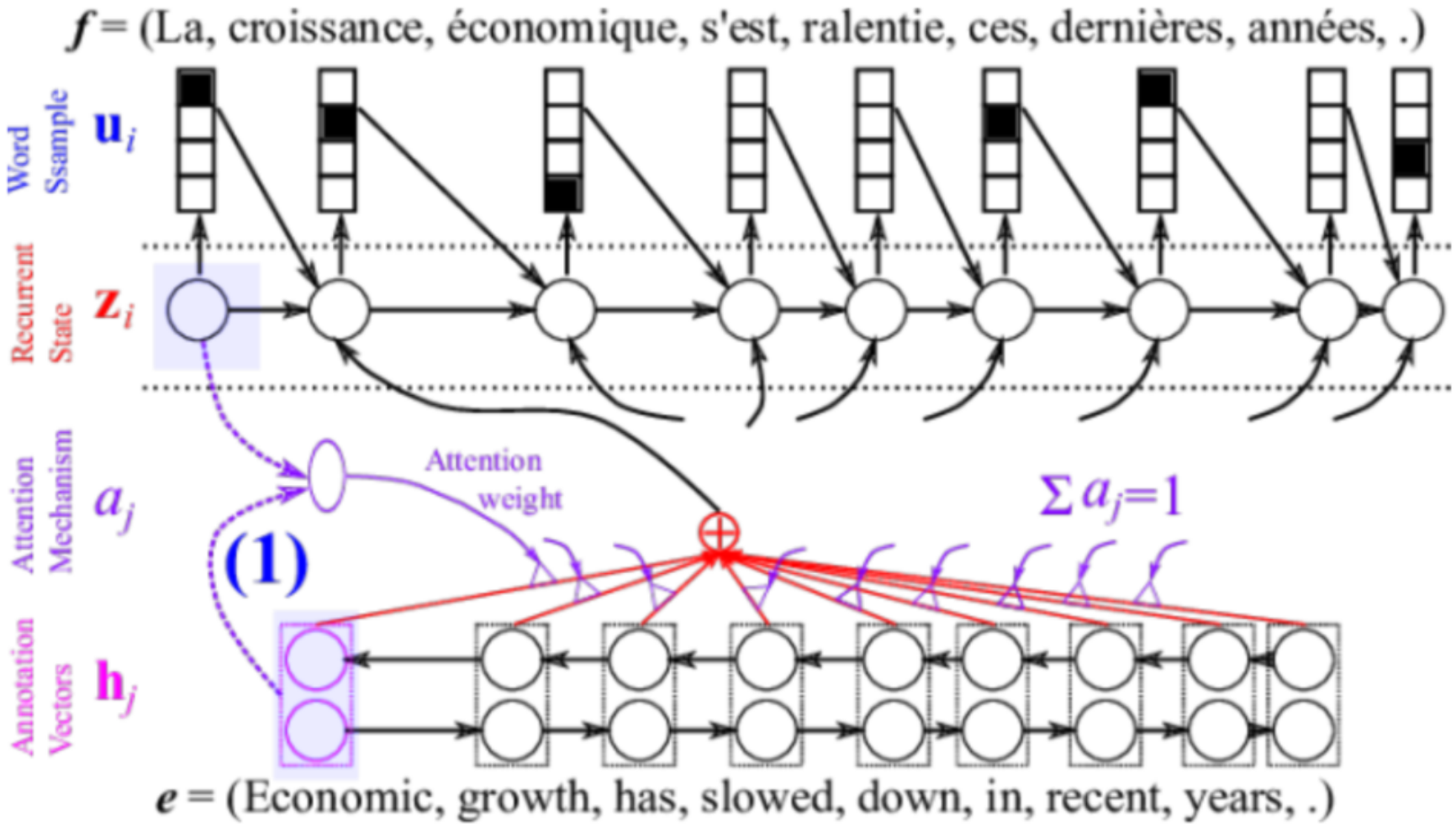

A **stop** sign is on a road with a mountain in the background

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

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

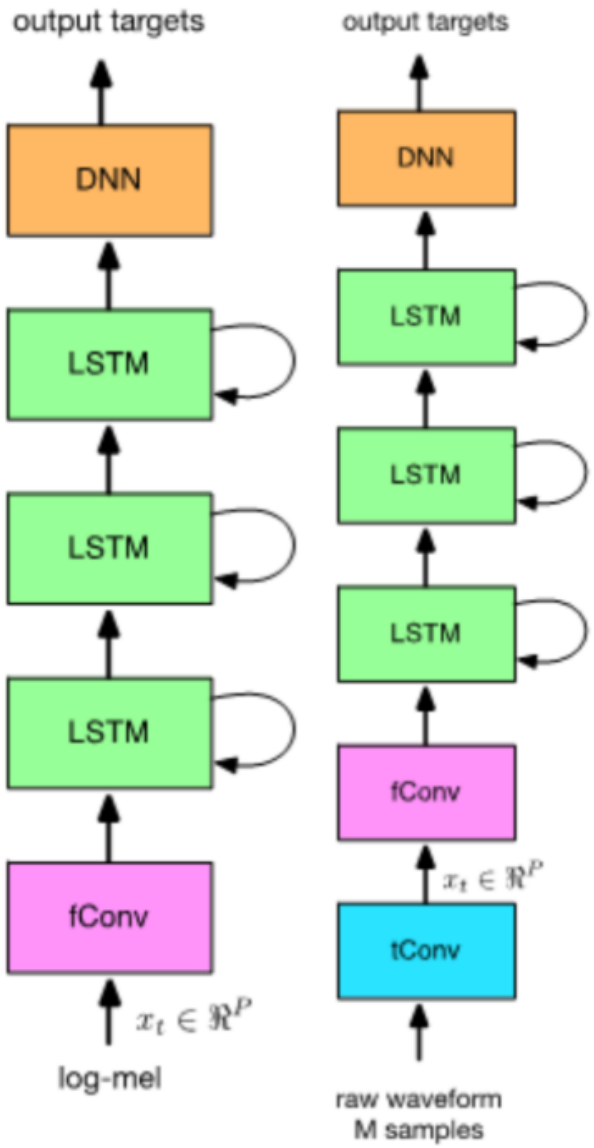
A giraffe standing in a forest with **trees** in the background.

# Example: Machine translation



# Example: Speech Recognition (Google CLDNN)

- **Feature learning**
  - CNN-based fConv layer
- **Temporal modeling**
  - LSTM layers
  - unroll 20 timesteps
- **Classification**
  - DNN layer (1024 ReLU)
  - 512 outputs

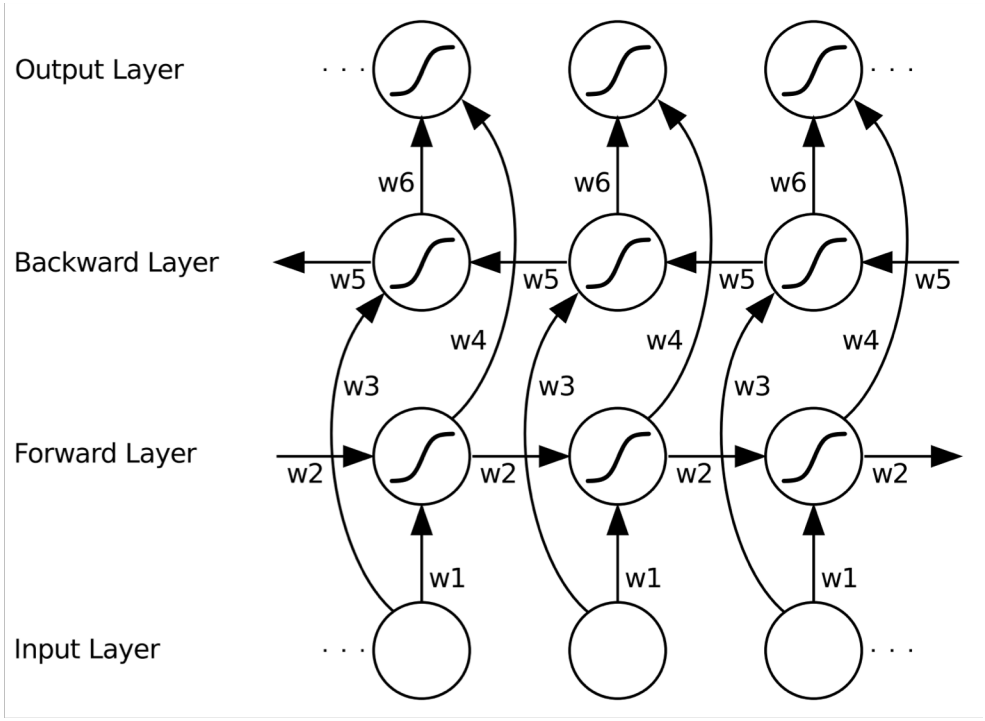


# Basic RNN training and augmentation with memory

- **RNNs are very powerful dynamic systems but**
  - training them has proved to be problematic
- **1980's: gradients-based fundamental methods**
  - backpropagation through time (BPTT)
  - real-time recurrent learning (RTRL)
- **1990's: proposals to augment RNNs with memory**
  - long short-term memory networks (Schmidhuber, 1997)
- **more recent approaches**
  - gated recurrent units (Cho, 2014)
  - neural turing machines (Graves and Danihelka, 2014)
  - memory networks (Weston and Bordes, 2014)

# Bidirectional RNN

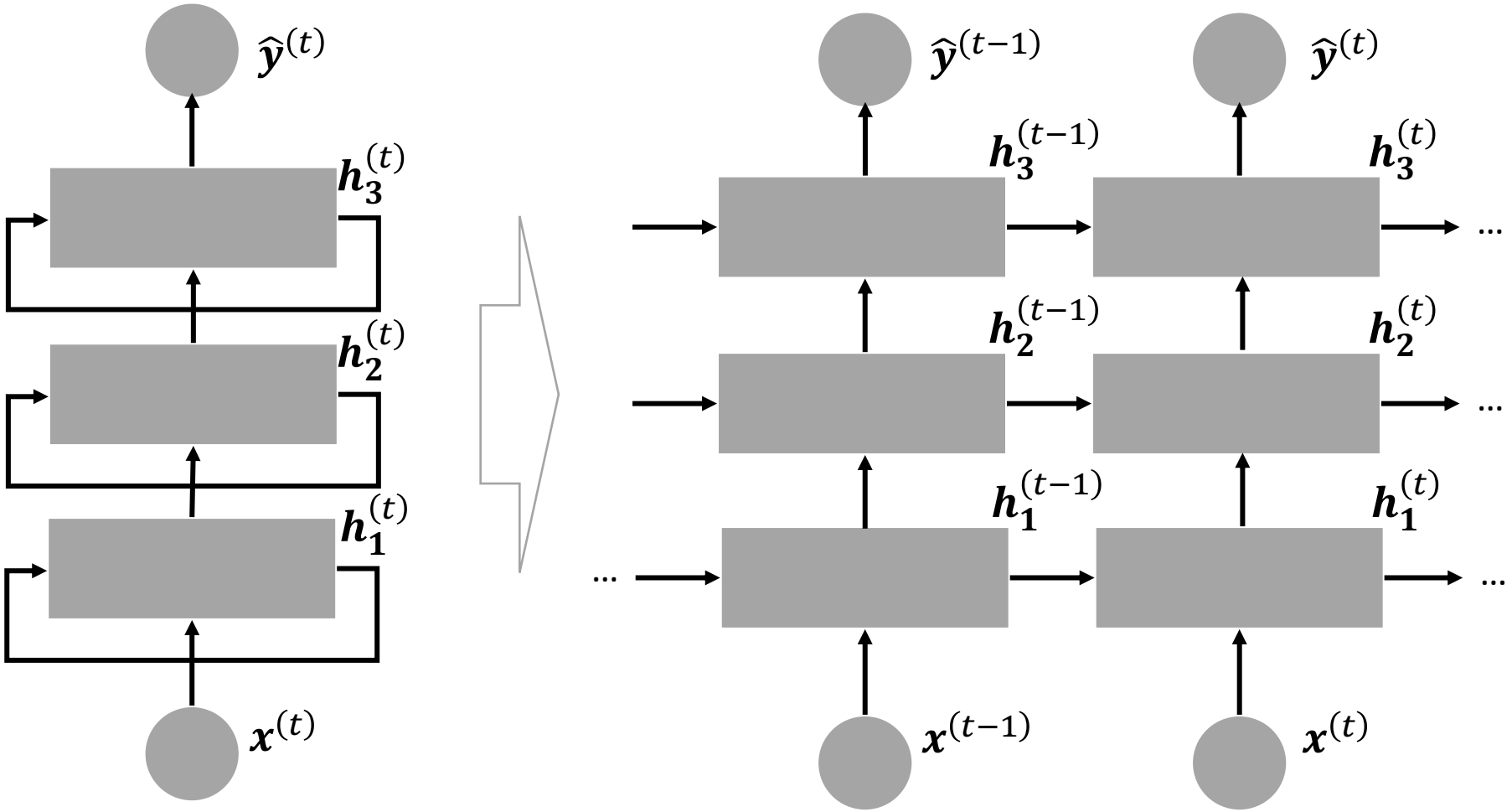
- **본래 순서의 시퀀스와 역순서의 시퀀스에 대한 각각의 RNN을 만든 후, 두 RNN의 출력층을 통합**
- **현재 시점의 출력값에 대해 과거와 미래의 state가 모두 반영됨**
  - 더 많은 정보가 이용된다는 측면에서, 본래 순서의 시퀀스만 사용한 RNN보다 더 좋은 효과가 있을 수 있음




Alex Graves, “Supervised sequence labeling with recurrent neural network”

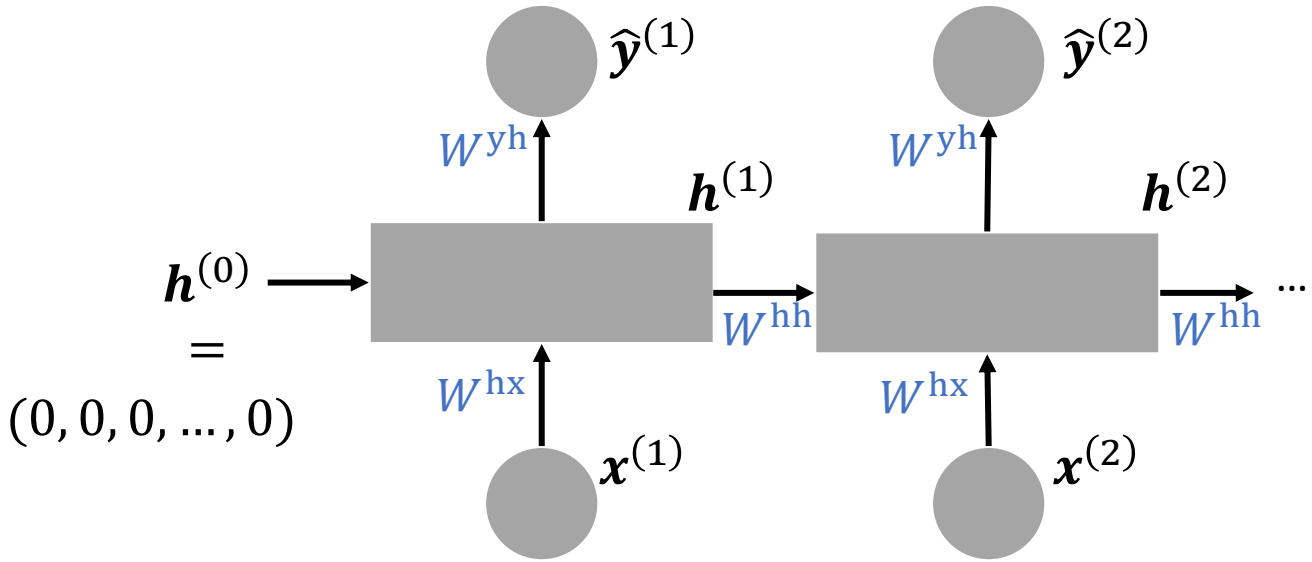
# Deep recurrent neural net

- 순환하는 은닉층을 다수 쌓음으로써 신경망구조를 더욱 깊게 만들 수 있음



# Initial state vector for RNNs

- **Generally, a zero state is used.**
  - However, there are some variations.
    - e.g., make trainable initial state vector.



# Backpropagation for RNN training

- **Goal: Determine each gradient to update weights w by stochastic gradient descent**

- **Well-know algorithms to calculate weight derivatives for RNN**
  - Backpropagation through time (BPTT)
  - Real-time recurrent learning(RTRL)

# Backpropagation through time (BPTT)

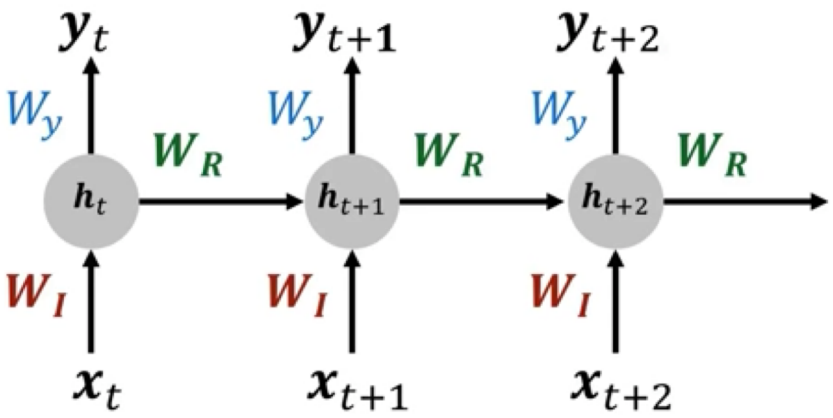
- **a gradient-based technique for training RNNs (Elman networks)**
  - Williams and Zipser (1995)
  - conceptually simpler and more efficient (in time) than RTRL
  - like standard backprop, BPTT consists of a repeated application of the chain rule
- **the loss function depends on the activation of the hidden layer**
  - not only through its influence on the output layer
  - but also through its influence on the hidden layer at the next timestep

# BPTT

- **Propagation**

$$\boldsymbol{h}^{(t)} = g_h(W_{\mathrm{I}}\boldsymbol{x}^{(t)} + W_{\mathrm{R}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h)$$

$$\boldsymbol{y}^{(t)} = g_y(W_y\boldsymbol{h}^{(t)} + \boldsymbol{b}_y)$$



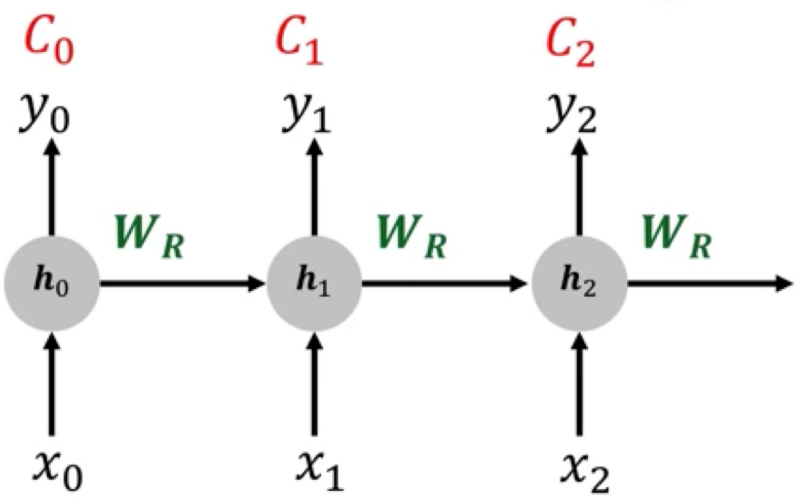

Nervana systems, https://www.youtube.com/watch?v=Ukgii7Yd_cU

# BPTT

- **Backpropagation**

$$\boldsymbol{h}^{(t)} = g_h(W_{\mathrm{I}}\boldsymbol{x}^{(t)} + W_{\mathrm{R}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h)$$

$$\boldsymbol{y}^{(t)} = g_y(W_y\boldsymbol{h}^{(t)} + \boldsymbol{b}_y)$$




Combine via:

$$\frac{\partial C}{\partial W_R} = \sum_t \frac{\partial C_t}{\partial W_R}$$

Example gradient:

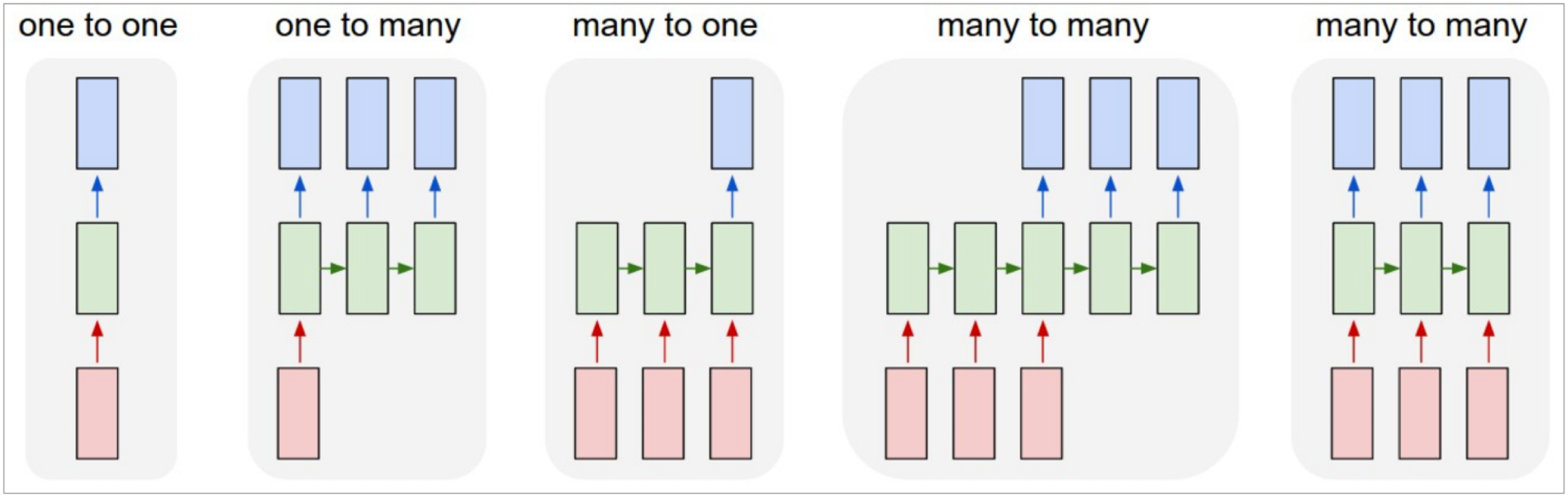
$$\frac{\partial C_2}{\partial W_R} = \frac{\partial C_2}{\partial y_2}\frac{\partial y_2}{\partial h_2}\frac{\partial h_2}{\partial g}\frac{\partial g}{\partial a}\frac{\partial a}{\partial W_R}$$

$$a = (W_{\mathrm{I}}x_2 + W_{\mathrm{R}}h_1 + b_h)$$

Depends on $W_R$ too!

Nervana systems, https://www.youtube.com/watch?v=Ukgii7Yd_cU

# Types of recurrent neural network




Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

# Contents

# Conventional RNN's limitation

- **RNNs can theoretically capture long-term dependencies**
  - but they are very hard to actually train to do this
- **RNNs have been found to perform better with**
  - more complex ("gated") activation units:
  - LSTM (long short-term memory): Hochreiter (1997)
  - GRU (gated recurrent unit): Cho (2014)
- **the main difference of GRU with LSTM**
  - In GRU, a single gating unit simultaneously controls the forgetting factor and the decision to update the state unit

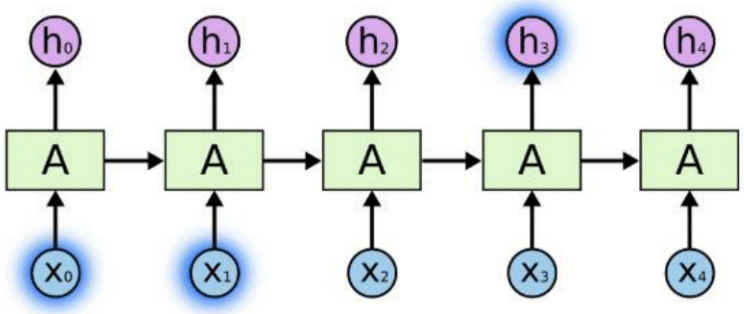
# Long short-term memory networks (LSTMs)

- **LSTM**
  - a recurrent neural network that is trained using back-propagation through time (BPTT) and overcomes the vanishing gradient problem.
  - As such it can be used to create large (stacked) recurrent networks, that in turn can be used to address difficult sequence problems in machine learning and achieve state-of-the-art results.
  - Instead of neurons, LSTM networks have memory blocks that are connected into layers.
  - A block contains gates that manage the block's state and output.

# Long short-term memory networks (LSTMs)

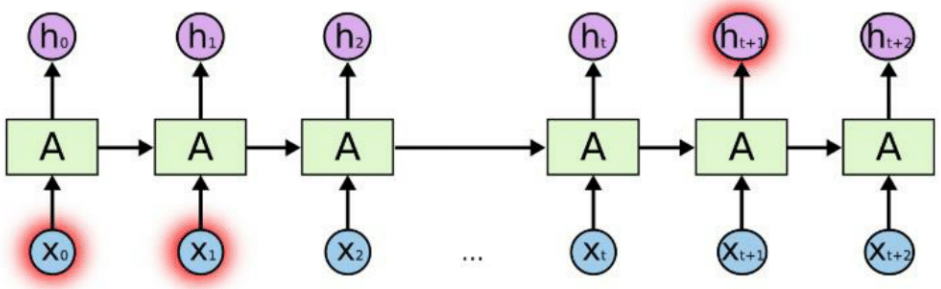
- **LSTM**
  - There are three types of gates within a memory unit:
    - Forget Gate: conditionally decides what information to discard from the unit.
    - Input Gate: conditionally decides which values from the input to update the memory state.
    - Output Gate: conditionally decides what to output based on input and the memory of the unit.

# Long-Term Dependency




- Short-term dependence:
  Bob is eating an **apple.**
- Long-term dependence:

Context → **Bob** likes **apples**. He is hungry and decided to have a snack. So now he is eating an **apple.**

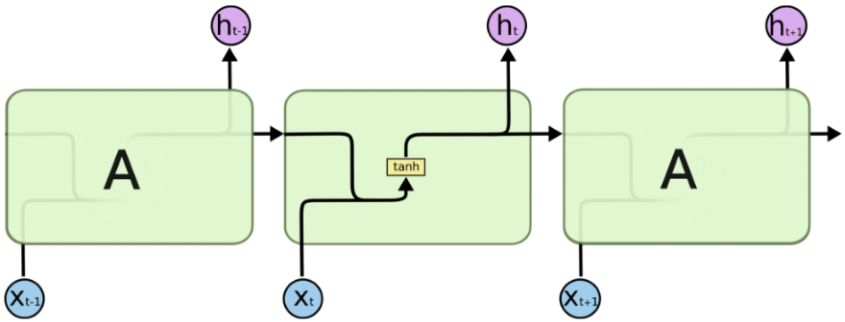



**In theory,** vanilla RNNs can handle arbitrarily long-term dependence.

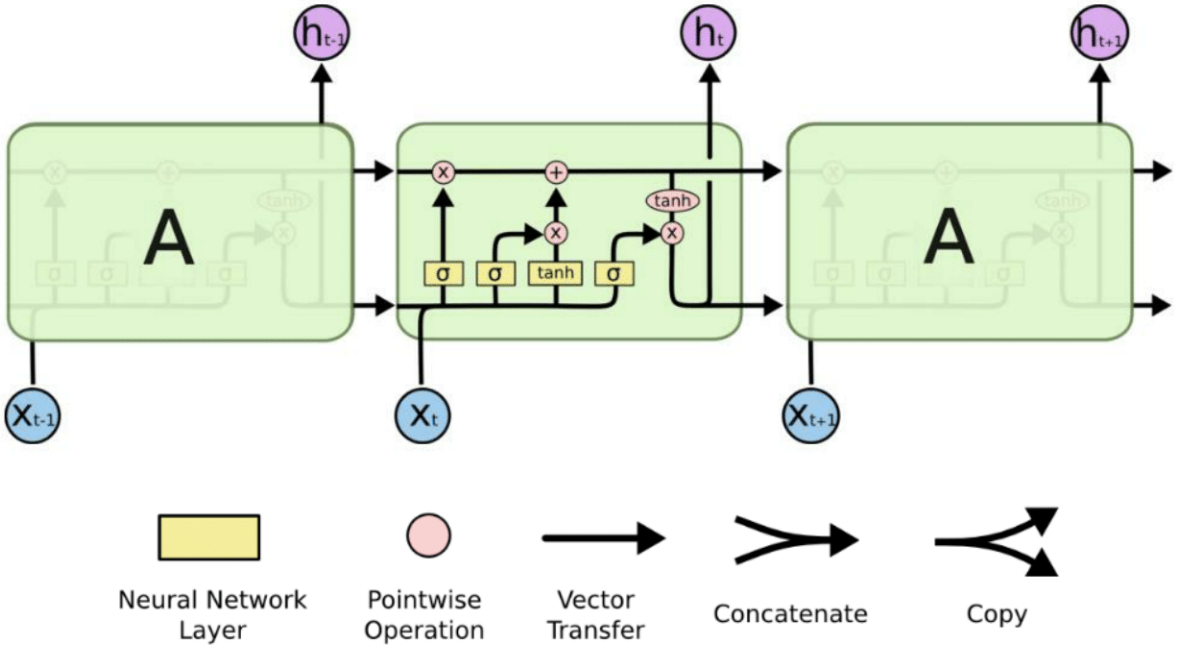
**In practice,** it's difficult.

# Long Short Term Memory (LSTM) Networks

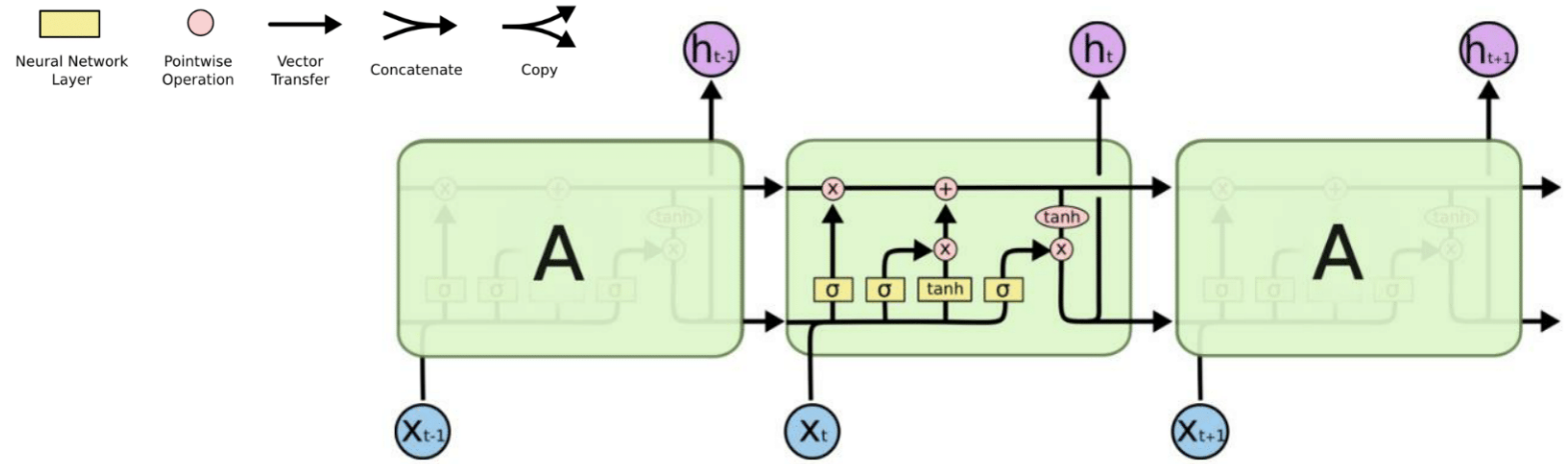
**Vanilla RNN:**




**LSTM:**



# LSTM: Pick What to Forget and What To Remember



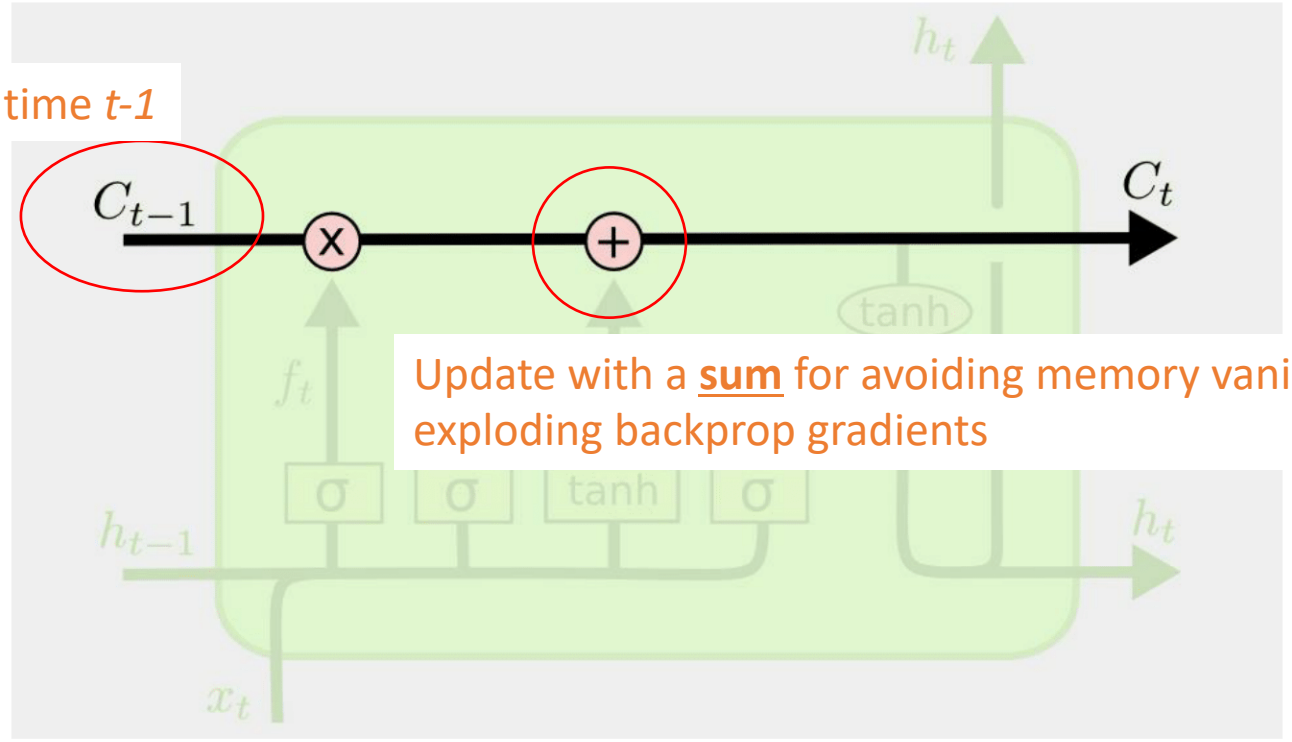

Bob and Alice are having lunch. Bob likes apples. Alice likes oranges.
**She is eating an orange.**

Conveyer belt for **previous state** and **new data**:

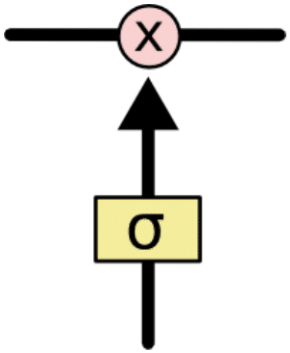
1. Decide what to forget (state)
2. Decide what to remember (state)
3. Decide what to output (if anything)

# LSTM Conveyer Belt

cell state at time *t-1*

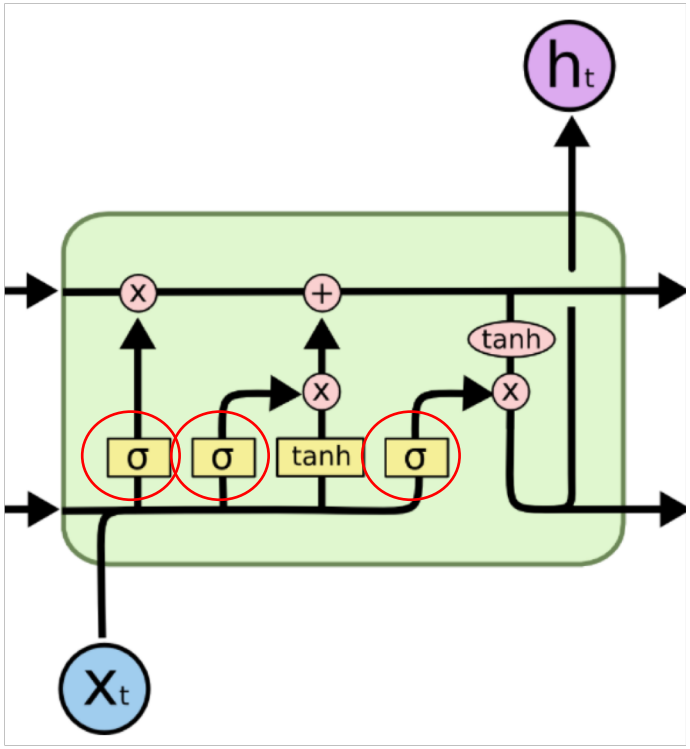


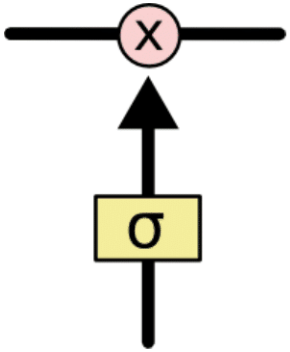

Update with a **sum** for avoiding memory vanishing and exploding backprop gradients




- State run through the cell
- 3 sigmoid layers output deciding which information is let through (~1) and which is not (~0)

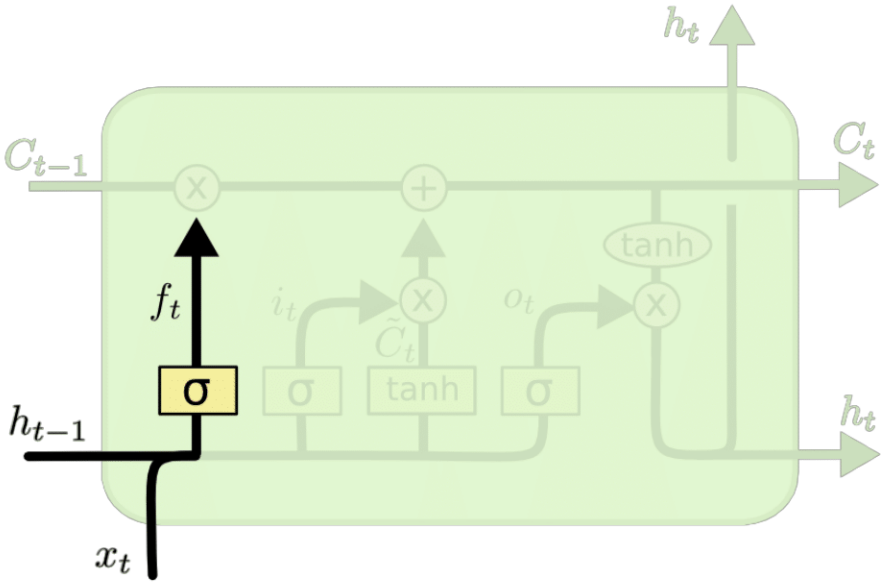
# LSTM Conveyer belt




- State run through the cell
- 3 sigmoid layers output deciding which information is let through (~1) and which is not (~0)

# LSTM Conveyer Belt

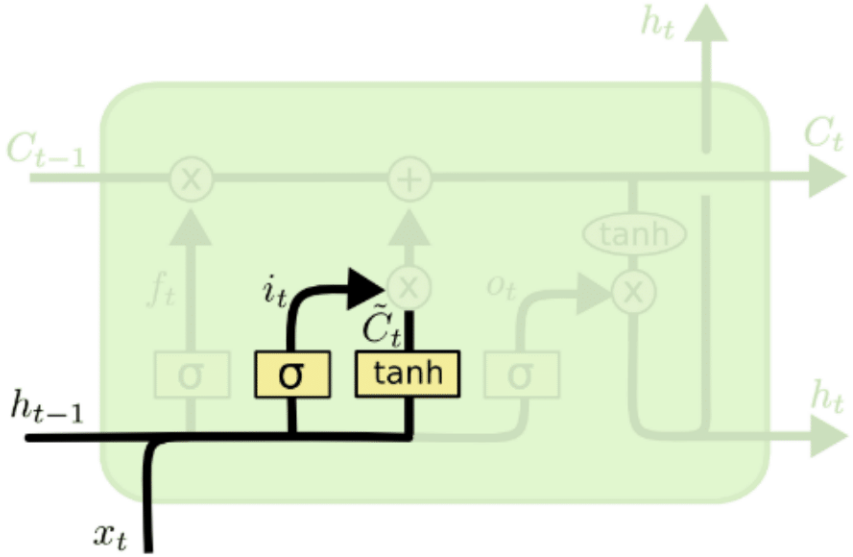



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

concatenate

**Step 1:** Decide what to forget / ignore

# LSTM Conveyer Belt

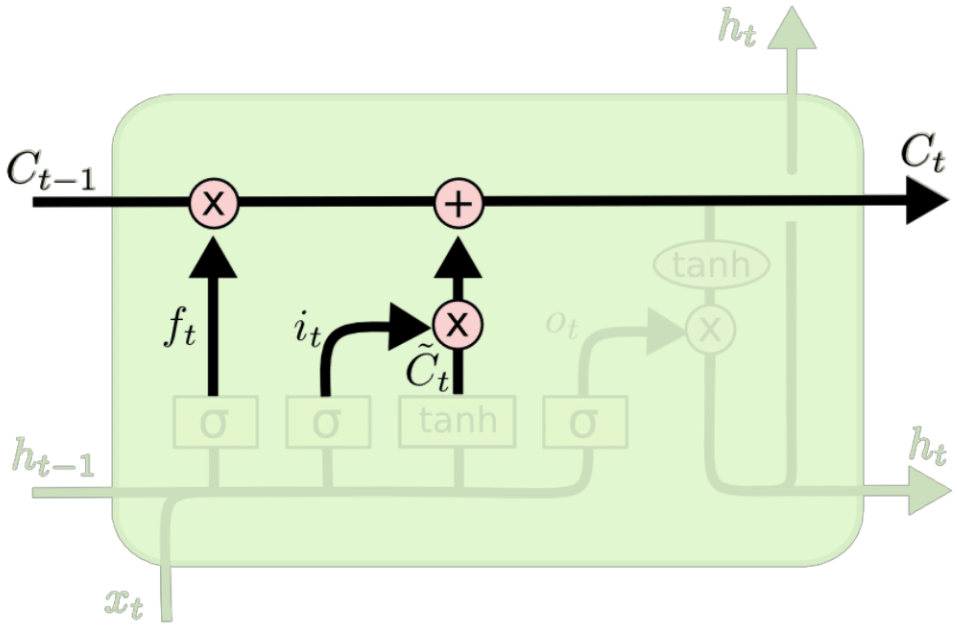



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

**Step 2:** Decide which state values to update (w/ sigmoid) and what values to update with (w/ tanh)

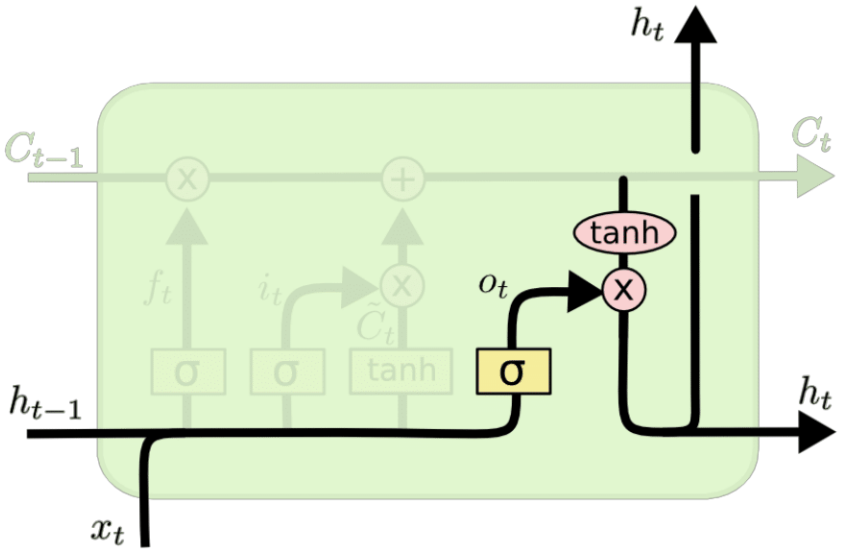
# LSTM Conveyer Belt




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

**Step 3:** Perform the forgetting and the state update

# LSTM Conveyer Belt




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

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

**Step 4:** Produce output with tanh [-1, 1] deciding the values and sigmoid [0, 1] deciding the filtering

# Gated recurrent units (GRUs)

- **main idea**
  - (a) have more persistent memory thereby making it easier for RNNs to capture long-term dependencies
  - (b) allow error to flow at different strengths depending on input

# Gated recurrent units (GRUs)

- **GRU computes the following at each time step.**
  - 1. update gate based on the current input and hidden state

$$z^t = \sigma\left(W_z x^t + U_z h^{t-1}\right)$$

  - 2. reset gate similarly (but with different weights)

$$r^t = \sigma\left(W_r x^t + U_r h^{t-1}\right)$$

  - 3. new memory content

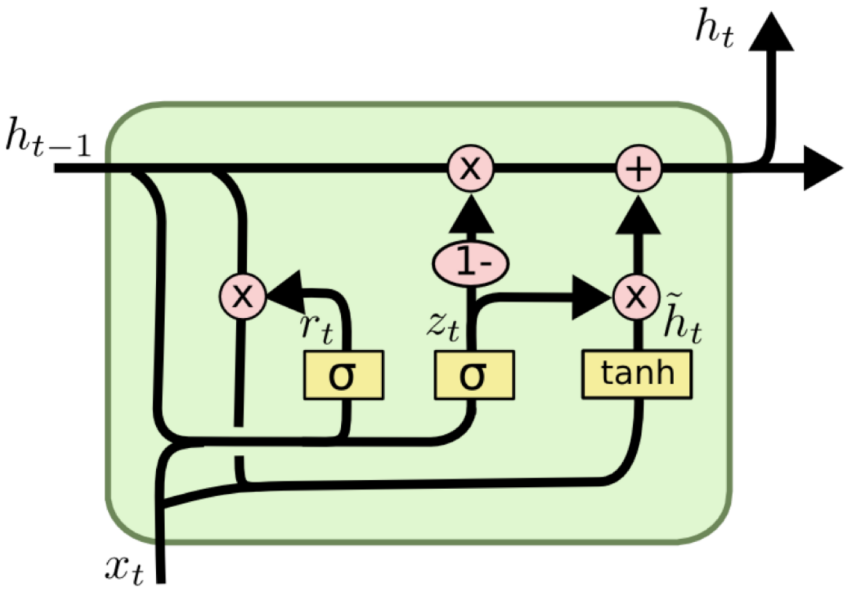
$$\tilde{h}^t = \tanh\left(W x^t + r^t \circ U h^{t-1}\right)$$

- **final memory combines current and previous time steps**

$$h^t = z^t \circ h^{t-1} + (1 - z^t) \circ \tilde{h}^t$$

# Gated recurrent units (GRUs)

- **Signal flow diagram**




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

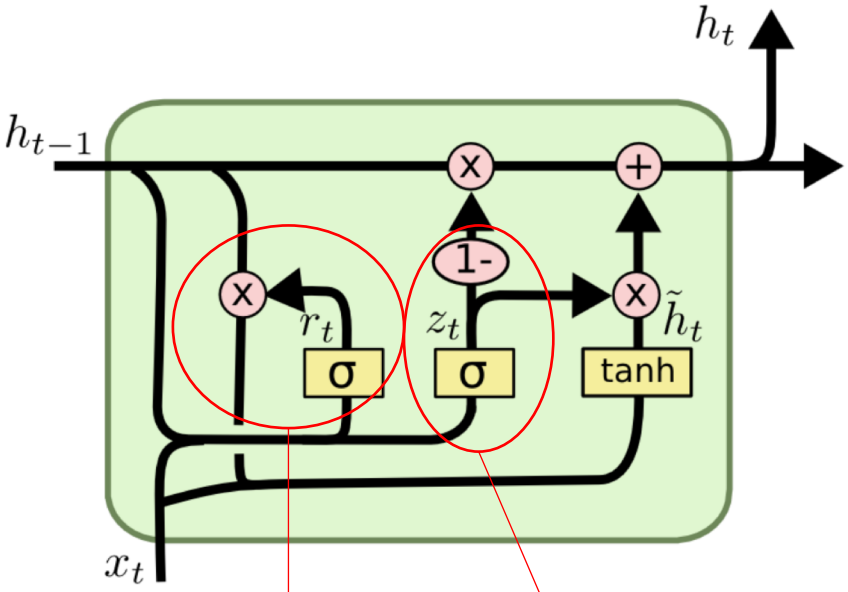
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

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

# Gated recurrent units (GRUs)

- **Signal flow diagram**




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

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

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

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

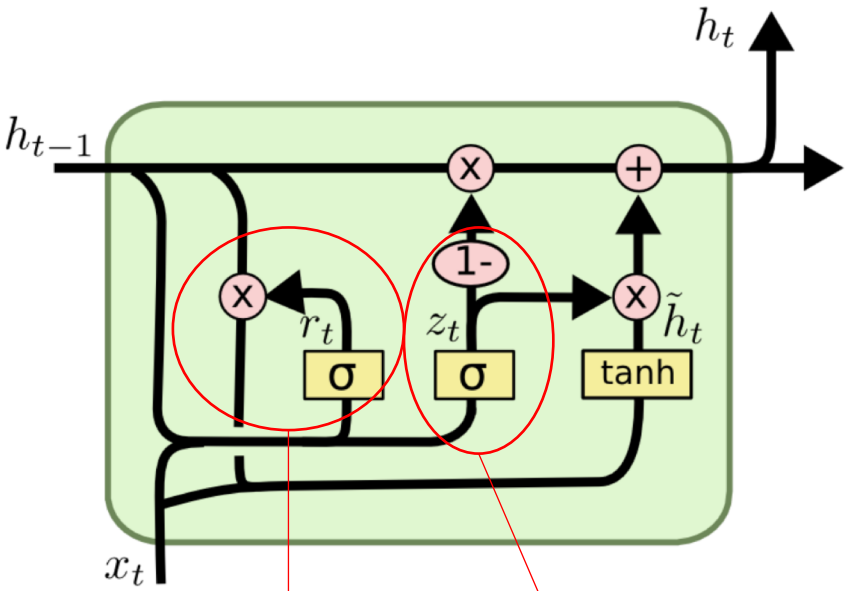
**reset gate**

**update gate**

# Gated recurrent units (GRUs)

- **Signal flow diagram**

If the reset gate *r* is closed to 0, the hidden state is forced to ignore the previous hidden state. ⇒ Any information which is found to be irrelevant later is ***dropped***.




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

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

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

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

**reset gate**

**update gate**

# Gated recurrent units (GRUs)

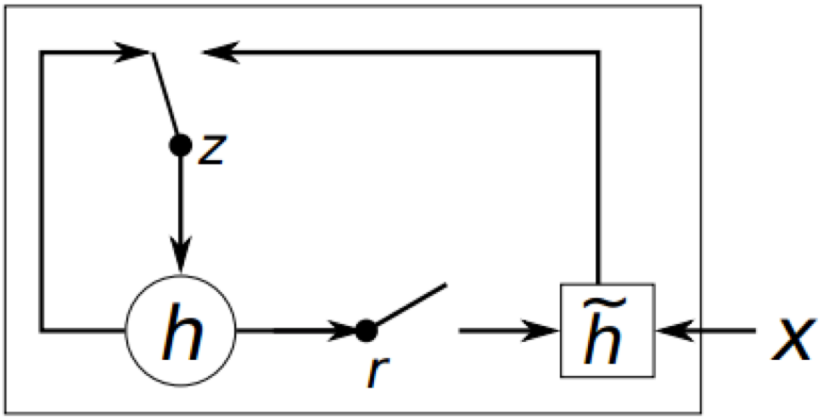
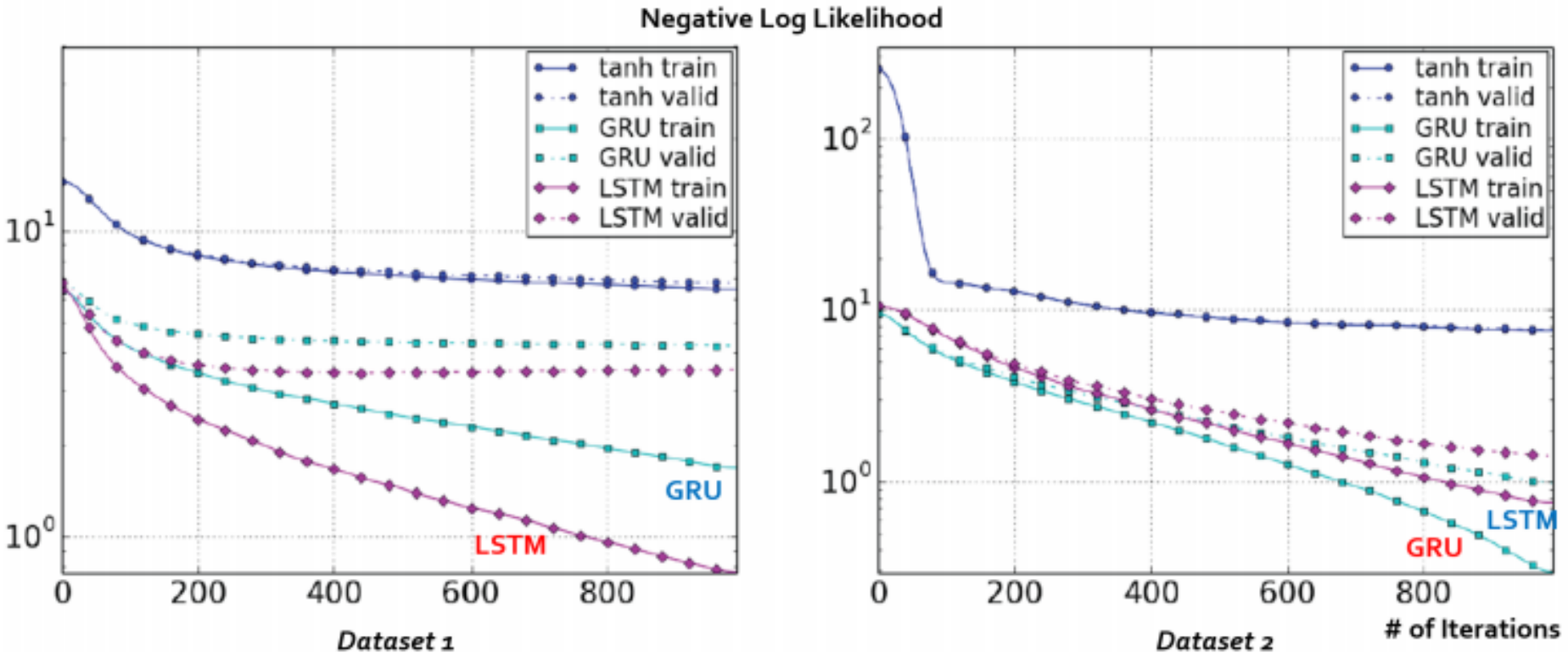



Figure 2: An illustration of the proposed hidden activation function. The update gate $z$ selects whether the hidden state is to be updated with a new hidden state $\tilde{h}$. The reset gate $r$ decides whether the previous hidden state is ignored. See Eqs. (5)–(8) for the detailed equations of $r$, $z$, $h$ and $\tilde{h}$.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

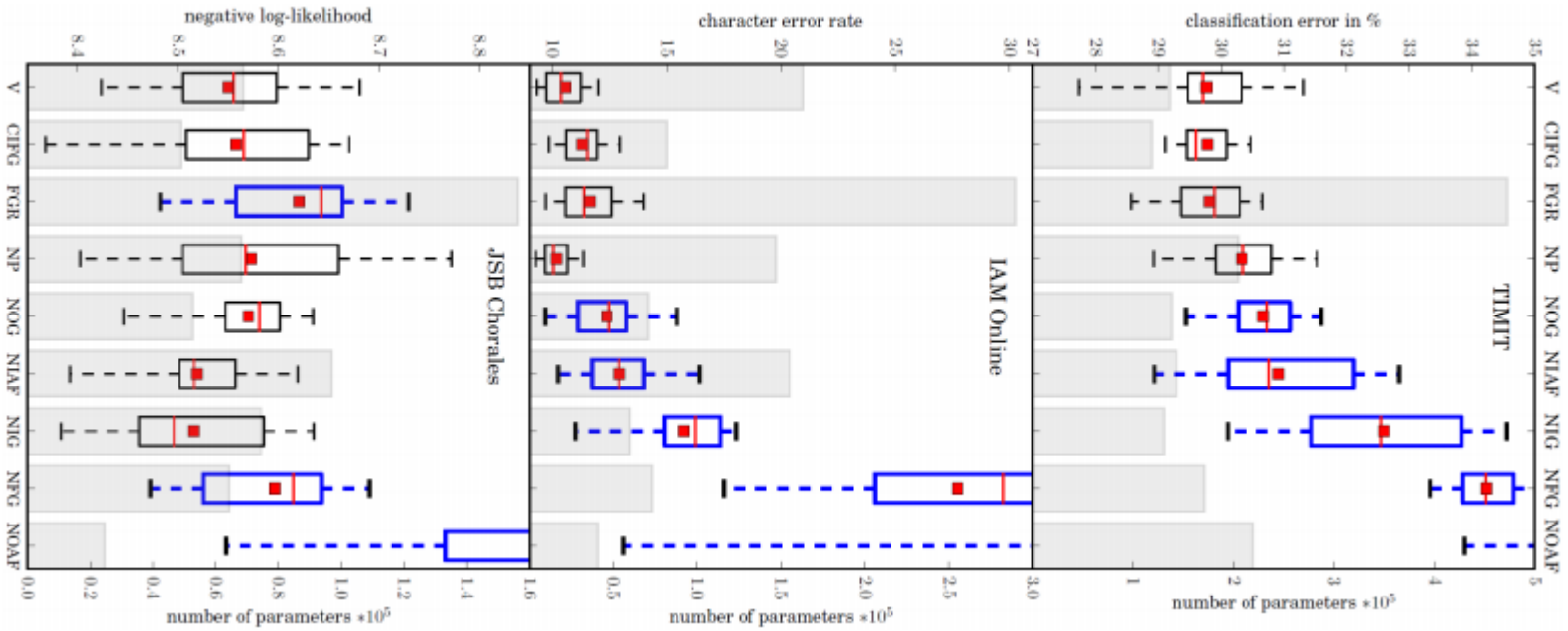
# “No clear winner" (Chung, 2014)

- **but GRU/LSTM-RNNs certainly outperform traditional RNNs**



# “LSTM: a search space odyssey" (Greff, 2016)

- **large-scale analysis of eight LSTM variants**
  - speech/handwriting recognition, polyphonic music modeling
  - 5400 experimental runs (15 years of CPU time)
- **result: “no variants can improve on standard LSTM significantly"**
  - most critical: forget gate & output activation function



# References


- 장병탁, 기계학습개론/딥러닝 강의노트 7장
- Christopher Olah's blog, "Understanding LSTM networks", http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Nervana systems, https://www.youtube.com/watch?v=Ukgii7Yd_cU
- MIT 6.S094: Deep learning for self-driving cars, Lecture 4, https://goo.gl/qG4Ys9
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.
- Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.
- Graves, A. (2012). Supervised sequence labelling. In *Supervised Sequence Labelling with Recurrent Neural Networks* (pp. 5-13). Springer Berlin Heidelberg.
- Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., & Schmidhuber, J. (2016). LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*.

# RNN-based language model

서울대학교병원 정보화실

고태훈 (taehoonko@snuh.org)

최세원 (swc@snuh.org)

# Language model

$$P(w_1, w_2, w_3, \ldots, w_m)$$

$$= \prod_{i=1}^{m} P(w_i | w_1, \ldots, w_{i-1})$$

# Recurrent neural network based language model

- **A simple recurrent neural network can be used to construct a language model.**

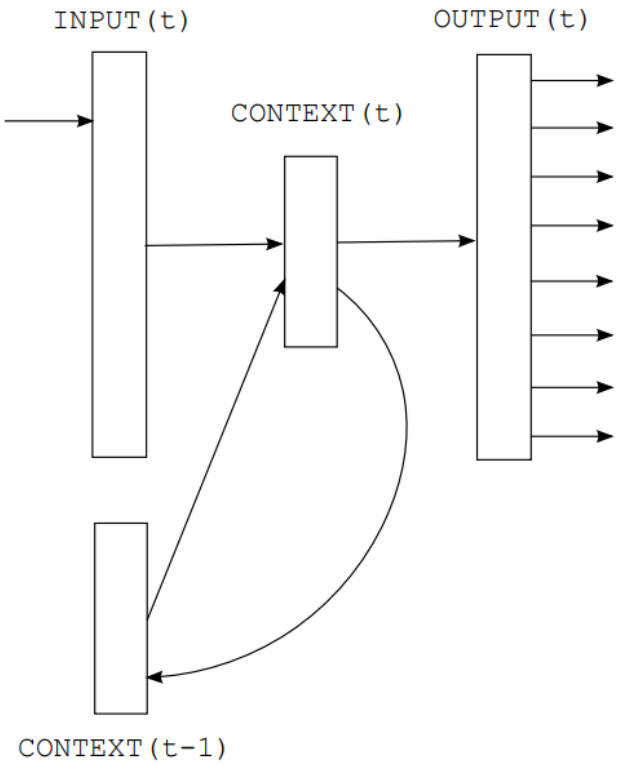



Figure 1: *Simple recurrent neural network.*

- $w(t)$: 1-of-N coding vector (word-level) at time t
- $s(t)$: state vector at time t
- $x(t) = w(t) \oplus s(t-1)$
- $s_j(t) = f(\sum_i x_i(t) u_{ji})$
- $y_k(t) = g(\sum_j s_j(t) v_{kj})$
  (where $f$ and $g$ are softmax functions)
- $y(t)$: probability distribution of next word given previous word $w(t)$ and state $s(t-1)$

(a.k.a. Elman network)

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., & Khudanpur, S. (2010, September). Recurrent neural network based language model. In *Interspeech*(Vol. 2, p. 3).

# Recurrent neural network based language model

- **A simple recurrent neural network can be used to construct a language model.**

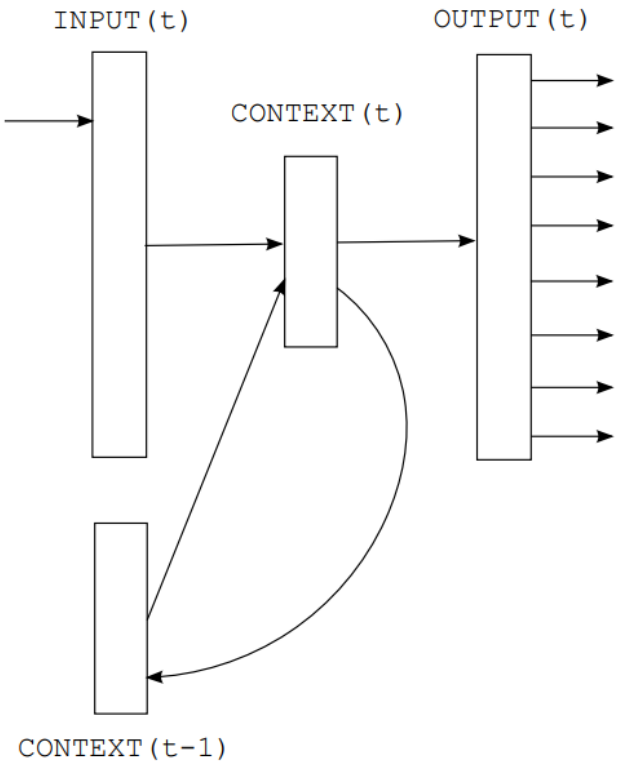



Figure 1: *Simple recurrent neural network.*

- Only 2 weight matrices are used.
- $s(t+1) = s(t)$ (They don't use weights).
- length of $s$: 30 ~ 500
- Stochastic gradient descent
- Starting learning rate = 0.1
  ➔ It is halved at start of each new epoch if there is no significant improvement.

(a.k.a. Elman network)

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., & Khudanpur, S. (2010, September). Recurrent neural network based language model. In *Interspeech*(Vol. 2, p. 3).

# Recurrent neural network based language model

- **Evaluation metric: *Perplexity***

$$perplexity = b^{\frac{H(p)}{n}}$$ Generally, $b = e \; or \; 2$.

  - $H(p)$: cross-entropy (or negative log-likelihood)
  - n: number of words in the input sequence

cross-entropy는 분류모델에서 주로 사용하는 평가지표이자 loss 함수이다. 그러나 cross-entropy는 단어 시퀀스의 길이를 고려하지 않는다.

Perplexity는 **단어 시퀀스가 관측된 시점에서 그 다음에 등장할 단어 후보의 개수**와 일맥상통하며, perplexity가 작을수록 해당 language model의 성능이 좋다는 것을 의미한다.

# Recurrent neural network based language model

- **Evaluation metric: *WER (word error rate)***

$$WER = \frac{S + D + I}{N}$$

- S is the number of substitutions,
- D is the number of deletions,
- I is the number of insertions and
- N is the number of words in the reference

WER (word error rate) 는 정답 시퀀스 대비 예측된 시퀀스가 얼마만큼의 차이를 나타내주는 지표. Word-level Levenshtein distance 라고 할 수 있음.

https://en.wikipedia.org/wiki/Word_error_rate
https://martin-thoma.com/word-error-rate-calculation/

# Recurrent neural network based language model

Table 1: *Performance of models on WSJ DEV set when increasing size of training data.*

| Model | # words | PPL | WER |
|---|---|---|---|
| KN5 LM | 200K | 336 | 16.4 |
| KN5 LM + RNN 90/2 | 200K | 271 | 15.4 |
| KN5 LM | 1M | 287 | 15.1 |
| KN5 LM + RNN 90/2 | 1M | 225 | 14.0 |
| KN5 LM | 6.4M | 221 | 13.5 |
| KN5 LM + RNN 250/5 | 6.4M | 156 | 11.7 |

Table 2: *Comparison of various configurations of RNN LMs and combinations with backoff models while using 6.4M words in training data (WSJ DEV).*

| | PPL | | WER | |
|---|---|---|---|---|
| Model | RNN | RNN+KN | RNN | RNN+KN |
| KN5 - baseline | - | 221 | - | 13.5 |
| RNN 60/20 | 229 | 186 | 13.2 | 12.6 |
| RNN 90/10 | 202 | 173 | 12.8 | 12.2 |
| RNN 250/5 | 173 | 155 | 12.3 | 11.7 |
| RNN 250/2 | 176 | 156 | 12.0 | 11.9 |
| RNN 400/10 | 171 | 152 | 12.5 | 12.1 |
| 3xRNN static | 151 | 143 | 11.6 | 11.3 |
| 3xRNN dynamic | 128 | 121 | 11.3 | 11.1 |

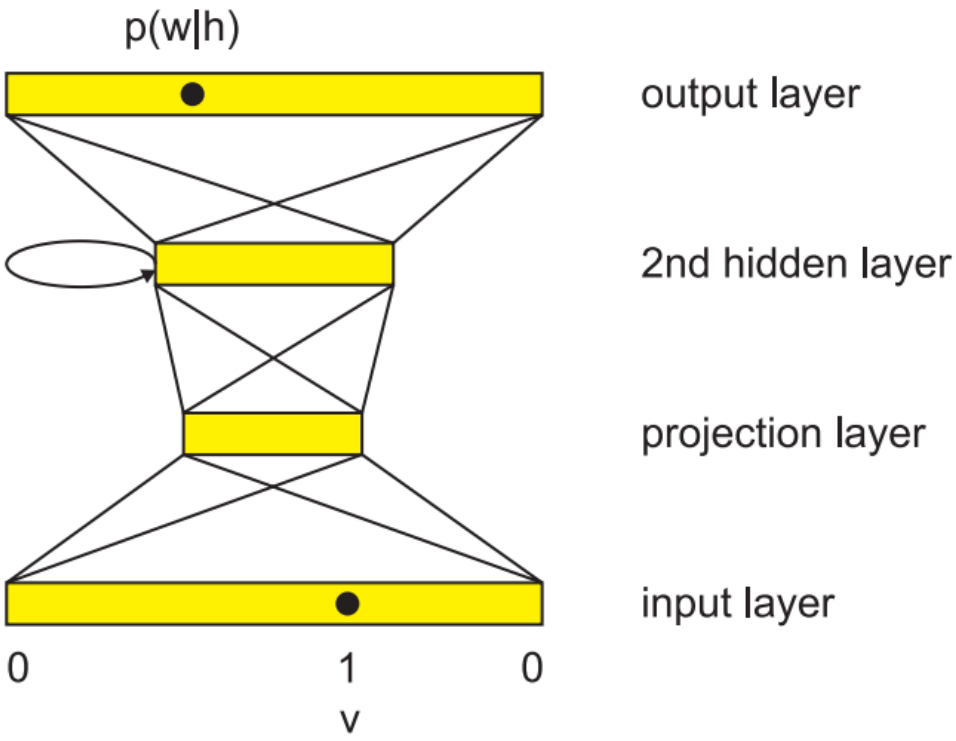
Table 3: *Comparison of WSJ results obtained with various models. Note that RNN models are trained just on 6.4M words.*

| Model | DEV WER | EVAL WER |
|---|---|---|
| Lattice 1 best | 12.9 | 18.4 |
| Baseline - KN5 (37M) | 12.2 | 17.2 |
| Discriminative LM [8] (37M) | 11.5 | 16.9 |
| Joint LM [9] (70M) | - | 16.7 |
| Static 3xRNN + KN5 (37M) | 11.0 | 15.5 |
| Dynamic 3xRNN + KN5 (37M) | 10.7 | 16.3[4] |

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., & Khudanpur, S. (2010, September). Recurrent neural network based language model. In *Interspeech*(Vol. 2, p. 3).

# LSTM neural networks for language modeling

- **Very similar to previous model except**
  - Projection layer: 1-of-N encoded vectors are mapped to a continuous space.
    (1-of-N encoded vector ➔ Embedded vector)
  - Using the LSTM layer




| LM | dev2 | test |
|---|---|---|
| KN 4-gram | 19.7 % | 17.6 % |
| KN 4-gram + LSTM | 19.2 % | 17.3 % |

Table 2: Word error rate results for Quaero French.

Sundermeyer, M., Schlüter, R., & Ney, H. (2012). LSTM neural networks for language modeling. In Thirteenth Annual Conference of the International Speech Communication Association.

# LSTM neural networks for language modeling

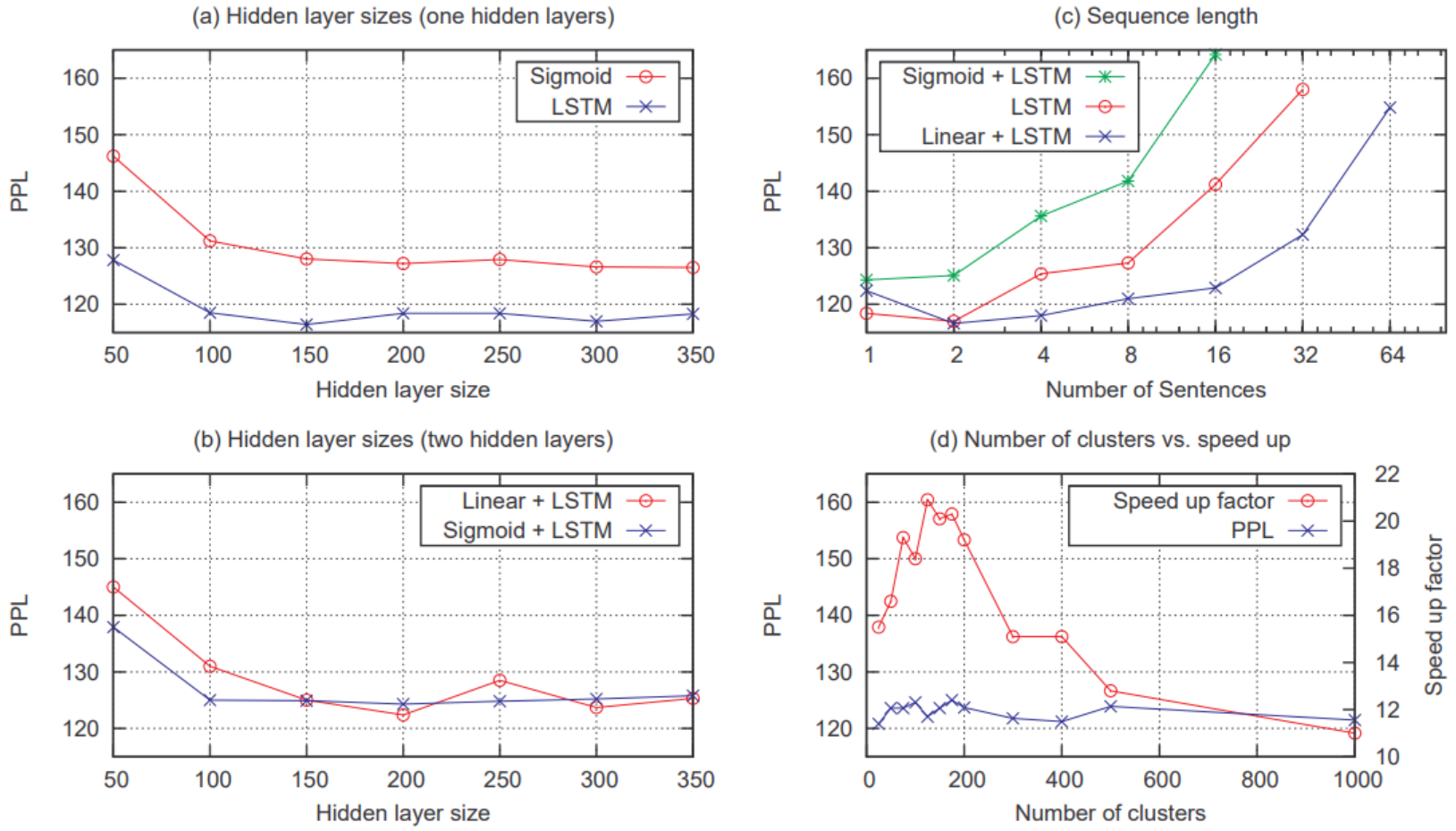
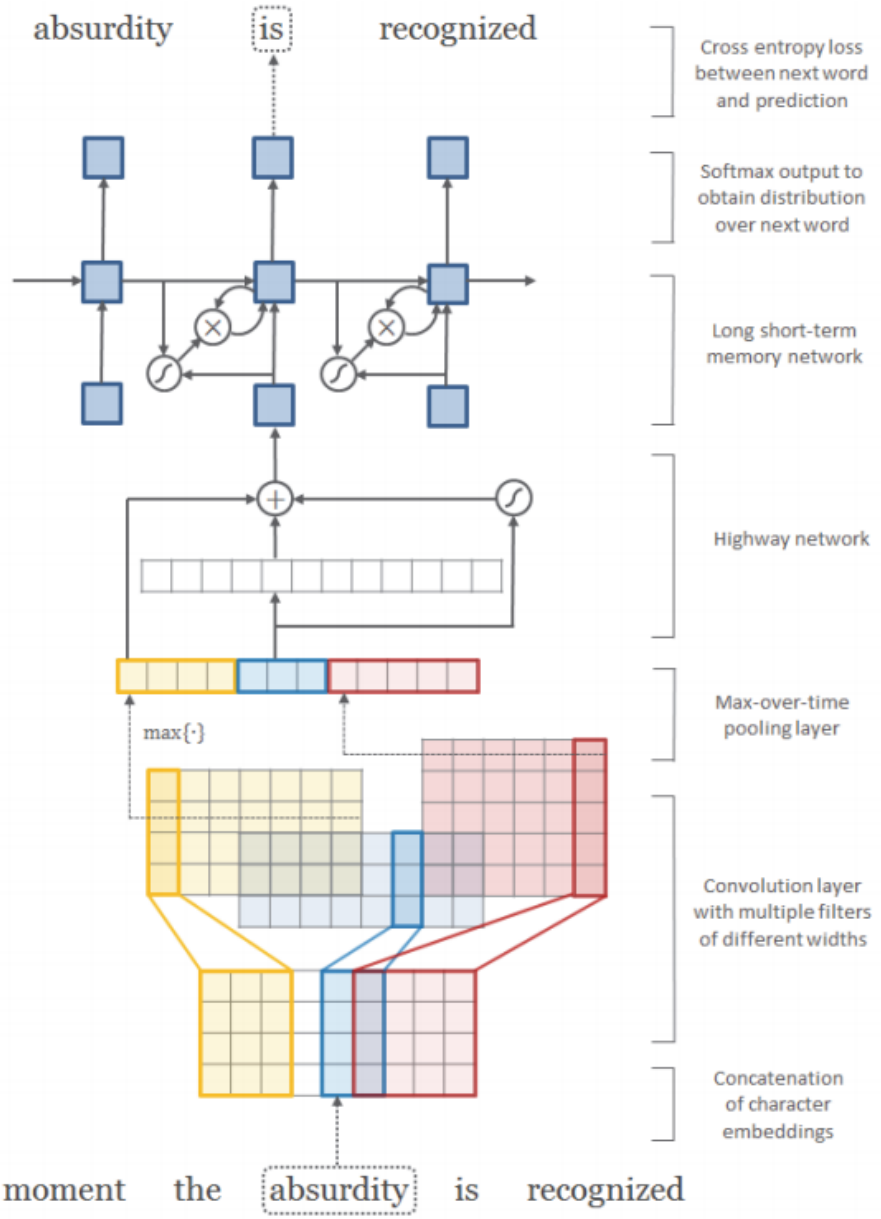



Figure 3: Experimental results on the Treebank corpus; for (c) and (d), 200 nodes were used for the hidden layers.

Sundermeyer, M., Schlüter, R., & Ney, H. (2012). LSTM neural networks for language modeling. In Thirteenth Annual Conference of the International Speech Communication Association.

# Character-aware neural language models

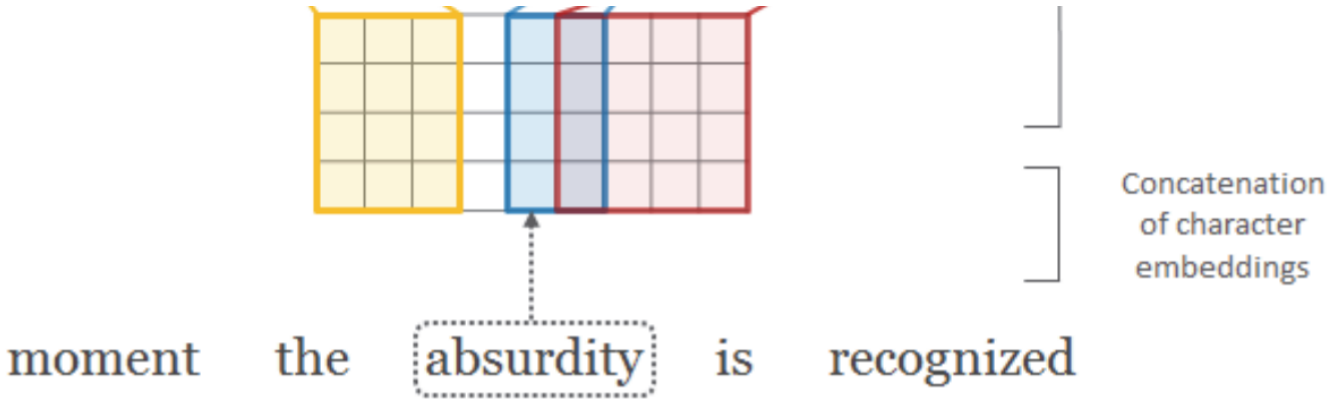
- **Char-level CNN**
  **+ highway network**
  **+ RNN-LM**




Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016, February). Character-Aware Neural Language Models. In AAAI (pp. 2741-2749).

# Character-aware neural language models

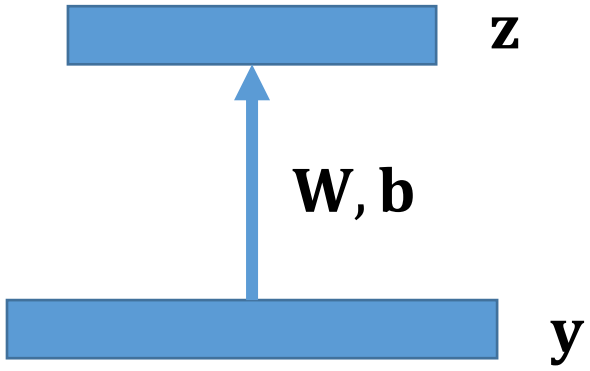
- **Char-level CNN**
  - Many researchers have used one-hot representations of characters.
  - But they found that using lower dimensional representations of characters performed slightly better if number of given characters is not large.

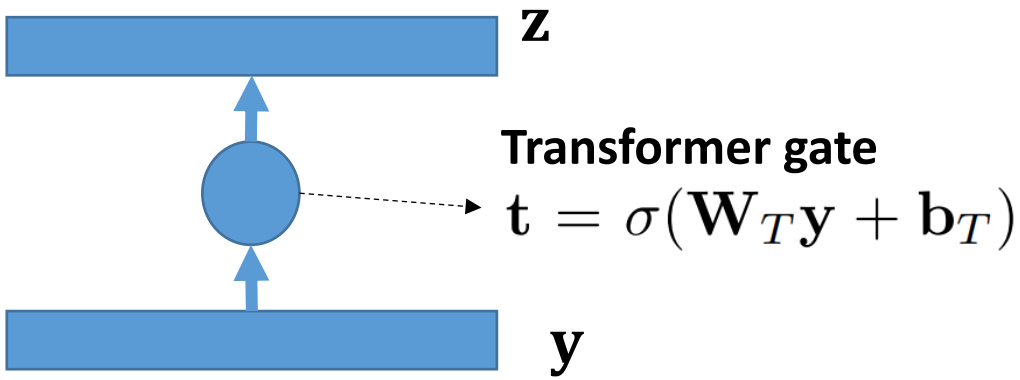



Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016, February). Character-Aware Neural Language Models. In AAAI (pp. 2741-2749).

# Character-aware neural language models

- **Highway network**
  - Highway network tried to increase the amount of information of extracted variables.
  - It **carries some dimensions of the input directly to the output**.
  - Very similar to the ResNet and the LSTM cell.




$$\mathbf{z} = g(\mathbf{W}\mathbf{y} + \mathbf{b})$$

$$\mathbf{z} = \mathbf{t} \odot g(\mathbf{W}_H\mathbf{y} + \mathbf{b}_H) + (\mathbf{1} - \mathbf{t}) \odot \mathbf{y}$$

Dimensions of **y** and **z** must be equal.

Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016, February). Character-Aware Neural Language Models. In AAAI (pp. 2741-2749).

# Character-aware neural language models

| | | Small | Large |
|---|---|---|---|
| CNN | $d$ | 15 | 15 |
| | $w$ | $[1, 2, 3, 4, 5, 6]$ | $[1, 2, 3, 4, 5, 6, 7]$ |
| | $h$ | $[25 \cdot w]$ | $[\min\{200, 50 \cdot w\}]$ |
| | $f$ | tanh | tanh |
| Highway | $l$ | 1 | 2 |
| | $g$ | ReLU | ReLU |
| LSTM | $l$ | 2 | 2 |
| | $m$ | 300 | 650 |

Table 2: Architecture of the small and large models. $d$ = dimensionality of character embeddings; $w$ = filter widths; $h$ = number of filter matrices, as a function of filter width (so the large model has filters of width $[1, 2, 3, 4, 5, 6, 7]$ of size $[50, 100, 150, 200, 200, 200, 200]$ for a total of 1100 filters); $f, g$ = nonlinearity functions; $l$ = number of layers; $m$ = number of hidden units.

| | $PPL$ | Size |
|---|---|---|
| LSTM-Word-Small | 97.6 | 5 m |
| LSTM-Char-Small | 92.3 | 5 m |
| LSTM-Word-Large | 85.4 | 20 m |
| LSTM-Char-Large | 78.9 | 19 m |
| KN-5 (Mikolov et al. 2012) | 141.2 | 2 m |
| RNN† (Mikolov et al. 2012) | 124.7 | 6 m |
| RNN-LDA† (Mikolov et al. 2012) | 113.7 | 7 m |
| genCNN† (Wang et al. 2015) | 116.4 | 8 m |
| FOFE-FNNLM† (Zhang et al. 2015) | 108.0 | 6 m |
| Deep RNN (Pascanu et al. 2013) | 107.5 | 6 m |
| Sum-Prod Net† (Cheng et al. 2014) | 100.0 | 5 m |
| LSTM-1† (Zaremba et al. 2014) | 82.7 | 20 m |
| LSTM-2† (Zaremba et al. 2014) | 78.4 | 52 m |

Table 3: Performance of our model versus other neural language models on the English Penn Treebank test set. $PPL$ refers to perplexity (lower is better) and size refers to the approximate number of parameters in the model. KN-5 is a Kneser-Ney 5-gram language model which serves as a non-neural baseline. †For these models the authors did not explicitly state the number of parameters, and hence sizes shown here are estimates based on our understanding of their papers or private correspondence with the respective authors.

RNN-LM with LSTM
+ Dropout

# Character-aware neural language models

| | LSTM-Char | |
|---|---|---|
| | Small | Large |
| No Highway Layers | 100.3 | 84.6 |
| One Highway Layer | 92.3 | 79.7 |
| Two Highway Layers | 90.1 | 78.9 |
| One MLP Layer | 111.2 | 92.6 |

Table 7: Perplexity on the Penn Treebank for small/large models trained with/without highway layers.

| | | $\|\mathcal{V}\|$ | | | |
|---|---|---|---|---|---|
| | | 10 k | 25 k | 50 k | 100 k |
| $T$ | 1 m | 17% | 16% | 21% | – |
| | 5 m | 8% | 14% | 16% | 21% |
| | 10 m | 9% | 9% | 12% | 15% |
| | 25 m | 9% | 8% | 9% | 10% |

Table 8: Perplexity reductions by going from small word-level to character-level models based on different corpus/vocabulary sizes on German (DE). $|\mathcal{V}|$ is the vocabulary size and $T$ is the number of tokens in the training set. The full vocabulary of the 1m dataset was less than 100k and hence that scenario is unavailable.

# Character-aware neural language models

- https://github.com/yoonkim/lstm-char-cnn
- https://github.com/carpedm20/lstm-char-cnn-tensorflow
- https://github.com/mkroutikov/tf-lstm-char-cnn

# References


- Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., & Khudanpur, S. (2010, September). Recurrent neural network based language model. In *Interspeech*(Vol. 2, p. 3).
- Sundermeyer, M., Schlüter, R., & Ney, H. (2012). LSTM neural networks for language modeling. In Thirteenth Annual Conference of the International Speech Communication Association.
- Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016, February). Character-Aware Neural Language Models. In AAAI (pp. 2741-2749).