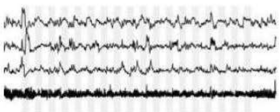
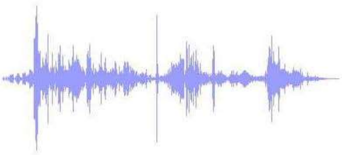
Recurrent Neural Networks

Sequence Modelling

What is a sequence?

- "This morning I took the dog for a walk." — sentence
-  — medical signals
-  — speech waveform

Why do we need RNNs?

The limitations of the Neural network (CNNs)

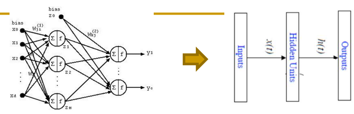
- Rely on the assumption of independence among the (training and test) examples.
  - After each data point is processed, the entire state of the network is lost
- Rely on examples being vectors of fixed length

To model sequences, we need:

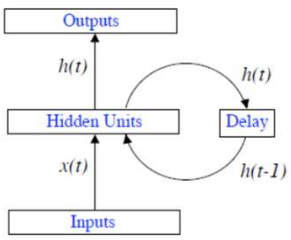
- To deal with variable-length sequences
- To maintain sequence order
- To keep track of long-term dependencies
- To share parameters across the sequence

Recurrent neural networks

What are RNNs?



Recurrent neural networks (RNNs) are connectionist models with the ability to selectively pass information across sequence steps, while processing sequential data one element at a time.



The simplest form of ***fully recurrent neural network*** is an MLP with the previous set of hidden unit activations feeding back into the network along with the inputs

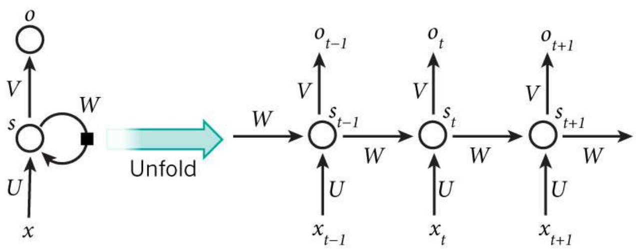
Allow a 'memory' of previous inputs to persist in the network's internal state, and thereby influence the network output

$$h(t) = f_H(W_{IH}x(t) + W_{HH}h(t-1))$$
$$y(t) = f_O(W_{HO}h(t))$$

f_H and f_O are the activation function for hidden and output unit; W_{IH}, W_{HH}, and W_{HO} are connection weight matrices which are learnt by training

What are RNNs?

- The recurrent network can be converted into a feed-forward network by ***unfolding over time***

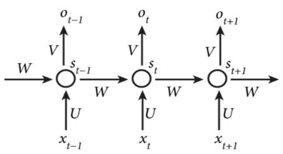


An unfolded recurrent network. Each node represents a layer of network units at a single time step. The weighted connections from the input layer to hidden layer are labelled 'U', those from the hidden layer to itself (i.e. the recurrent weights) are labelled 'W' and the hidden to output weights are labelled'V'. Note that the same weights are reused at every time step. Bias weights are omitted for clarity.

What are RNNs?

- Training RNNs (determine the parameters)

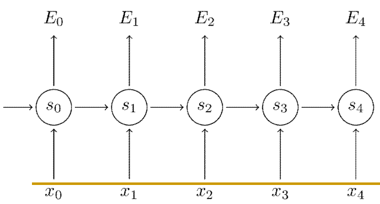
Back Propagation Through Time (BPTT) is often used to learn the RNN
BPTT is an extension of the back-propagation (BP)



- The output of this RNN is $\hat{y}_t$

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = softmax(Vs_t)$$

- The loss/error function of this network is



$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$ → The error at each time step

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$ → the total loss is the sum of the errors at each time step

What are RNNs?

- Training RNNs (determine the parameters)

✓ The gradients of the error with respect to our parameters

Just like we sum up the errors, we also sum up the gradients at each time step for one training example. For parameter W, the gradient is

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

✓ The gradient at each time step

we use time 3 as an example

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W} \longrightarrow \boxed{\text{Chain Rule}}$$

$$s_3 = \tanh(Ux_1 + Ws_2) \longrightarrow \boxed{s_3 \text{ depends on } W, s_2(W), s_1(W), s_0(W)}$$

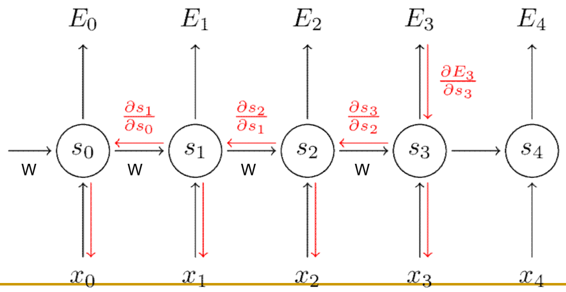
$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W} \longrightarrow \boxed{\text{Apply Chain Rule again on } s_k}$$

What are RNNs?

- Training RNNs (determine the parameters)

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

Becaise W is used in every step up to the output we care about, we need to back-propagate gradients from $t = 3$ through the network all the way to $t = 0$



What are RNNs?

Reminder:
$s_t = \sigma(a_t), \quad a_t = Ux_t + Ws_{t-1}$
$\frac{\partial s_t}{\partial s_{t-1}} = W^T \frac{\partial s_t}{\partial a_t} = W^T \sigma'(a_t)$

- The vanishing gradient problem

To understand why, let's take a closer look at the gradient we calculated above:

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W} \longrightarrow \frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \underbrace{\prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}}} \frac{\partial s_k}{\partial W}$$

Because the layers and time steps of deep neural networks relate to each other through multiplication, derivatives are susceptible to vanishing

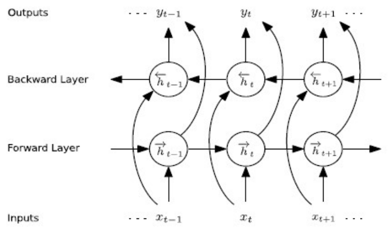
Gradient contributions from “far away” steps become zero, and the state at those steps doesn’t contribute to what you are learning: You end up not learning long-range dependencies.

What are RNNs?

- How to solve the vanishing gradient problem?
  - Proper initialization of the W matrix can reduce the effect of vanishing gradients
  - Use ReLU instead of tanh or sigmoid activation function
    ReLU derivate is a constant of either 0 or 1, so it isn’t likely to suffer from vanishing gradients
  - Use Long Short-Term Memory or Gated Recurrent unit architectures
    LSTM will be introduced later

RNN Extensions: Bidirectional Recurrent Neural Networks

Traditional RNNs only model the dependence of the current state on the previous state, BRNNs (*Schuster* and *Paliwal*, 1997) extend to model dependence on both past states and future states.
For example: predicting a missing word in a sequence you want to look at both the left and the right context.



An unfolded BRNN

$$\overrightarrow{h}_t = f(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}})$$
$$\overleftarrow{h}_t = f(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t-1} + b_{\overleftarrow{h}})$$

training sequence forwards and backwards to two separate recurrent hidden layers

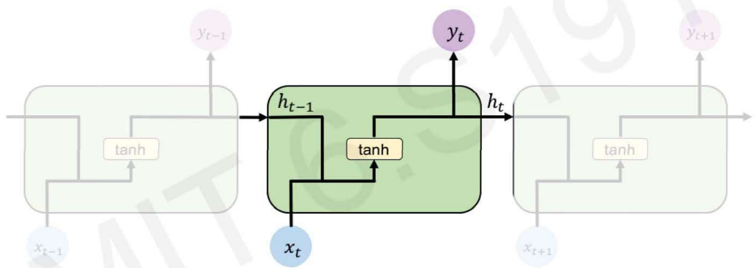
$$y_t = \underbrace{W_{\overrightarrow{h}y}\overrightarrow{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t}_{} + b_y$$

past and future context determines the output

RNN Extensions: Long Short-term Memory

Standard RNN

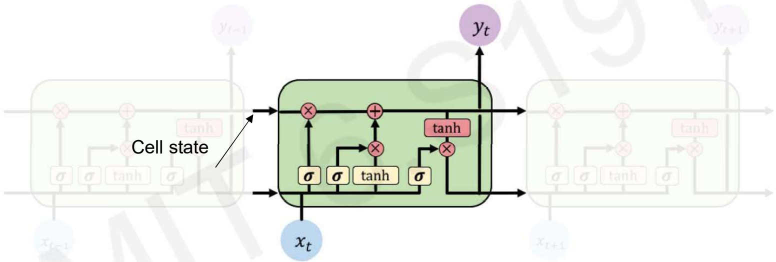
In a standard RNN, repeating modules contain a **simple computation node**



RNN Extensions: Long Short-term Memory

The vanishing gradient problem prevents standard RNNs from learning long-term dependencies. LSTMs (Hochreiter and Schmidhuber, 1997) were designed to combat vanishing gradients through a *gating* mechanism.

LSTM modules contain **computational blocks** that **control information flow**

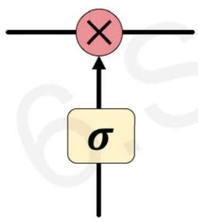


LSTM cells are able to track information throughout many timesteps

RNN Extensions: Long Short-term Memory

The vanishing gradient problem prevents standard RNNs from learning long-term dependencies. LSTMs (Hochreiter and Schmidhuber, 1997) were designed to combat vanishing gradients through a *gating* mechanism.

Information is **added** or **removed** through structures called **gates**



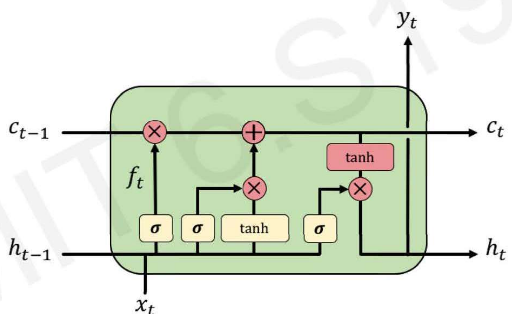
Gates optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication

RNN Extensions: Long Short-term Memory

The vanishing gradient problem prevents standard RNNs from learning long-term dependencies. LSTMs (Hochreiter and Schmidhuber, 1997) were designed to combat vanishing gradients through a *gating* mechanism.

How do LSTMs work?

1) Forget 2) Store 3) Update 4) Output

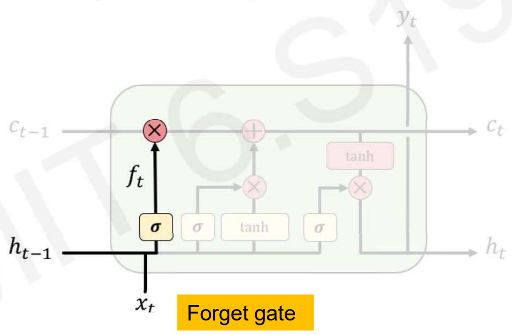


RNN Extensions: Long Short-term Memory

The vanishing gradient problem prevents standard RNNs from learning long-term dependencies. LSTMs (Hochreiter and Schmidhuber, 1997) were designed to combat vanishing gradients through a *gating* mechanism.

1) Forget 2) Store 3) Update 4) Output

LSTMs **forget irrelevant** parts of the previous state

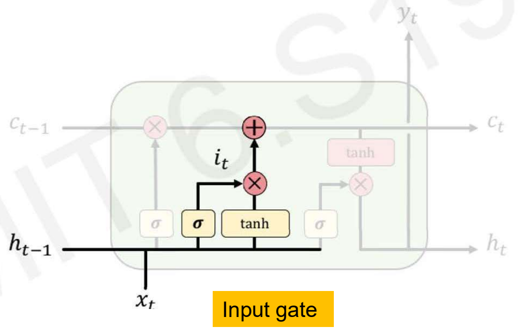


RNN Extensions: Long Short-term Memory

The vanishing gradient problem prevents standard RNNs from learning long-term dependencies. LSTMs (Hochreiter and Schmidhuber, 1997) were designed to combat vanishing gradients through a *gating* mechanism.

1) Forget **2) Store** 3) Update 4) Output

LSTMs **store relevant** new information into the cell state

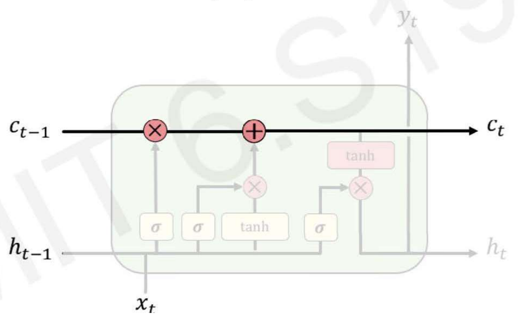


RNN Extensions: Long Short-term Memory

The vanishing gradient problem prevents standard RNNs from learning long-term dependencies. LSTMs (Hochreiter and Schmidhuber, 1997) were designed to combat vanishing gradients through a *gating* mechanism.

1) Forget 2) Store **3) Update** 4) Output

LSTMs **selectively update** cell state values

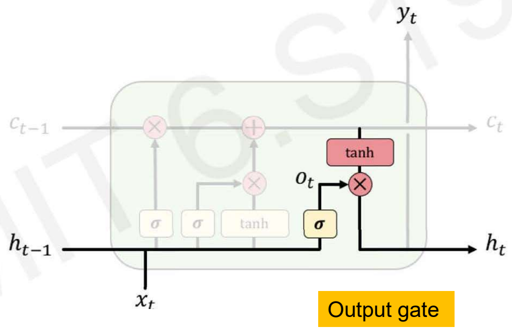


RNN Extensions: Long Short-term Memory

The vanishing gradient problem prevents standard RNNs from learning long-term dependencies. LSTMs (Hochreiter and Schmidhuber, 1997) were designed to combat vanishing gradients through a *gating* mechanism.

1) Forget 2) Store 3) Update **4) Output**

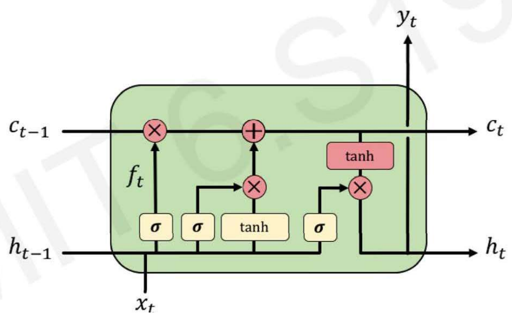
The **output gate** controls what information is sent to the next time step



RNN Extensions: Long Short-term Memory

The vanishing gradient problem prevents standard RNNs from learning long-term dependencies. LSTMs (Hochreiter and Schmidhuber, 1997) were designed to combat vanishing gradients through a *gating* mechanism.

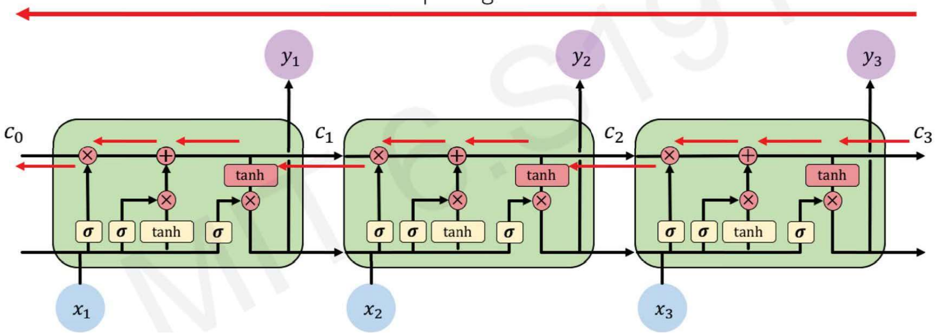
1) Forget 2) Store 3) Update 4) Output



RNN Extensions: Long Short-term Memory

LSTM Gradient Flow

Uninterrupted gradient flow!



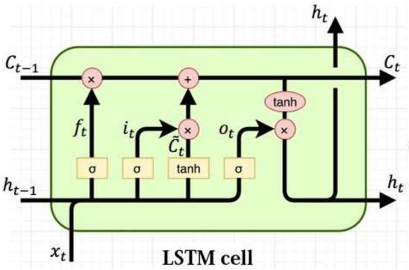
RNN Extensions: Long Short-term Memory

LSTMs: Key Concepts

1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
    - **Forget** gate gets rid of irrelevant information
    - **Store** relevant information from current input
    - Selectively **update** cell state
    - **Output** gate returns a filtered version of the cell state
3. Backpropagation through time with **uninterrupted gradient flow**

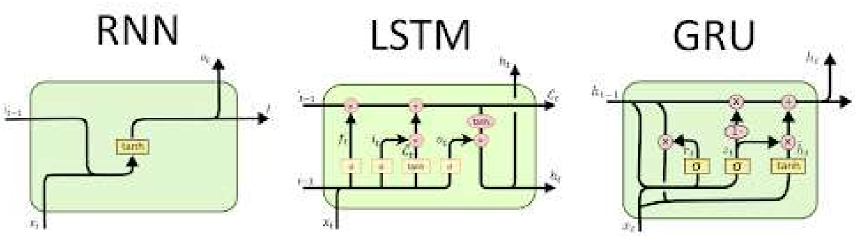
RNN extensions: Long Short-term Memory

LSTMs help preserve the error that can be back-propagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps (over 1000), thereby opening a channel to link causes and effects remotely



$$i_t = \sigma(x_t U^i + h_{t-1} W^i)$$
$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$
$$o_t = \sigma(x_t U^o + h_{t-1} W^o)$$
$$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$$
$$C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t)$$
$$h_t = \tanh(C_t) * o_t$$

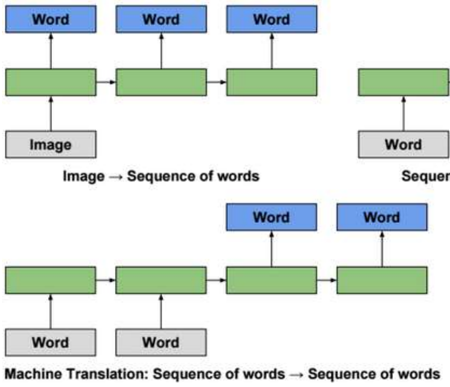
A comparison of three RNN models



RNN: Recurrent Neural Network
LSTM: Long Short-term Memory
GRU: Gated Recurrent Unit

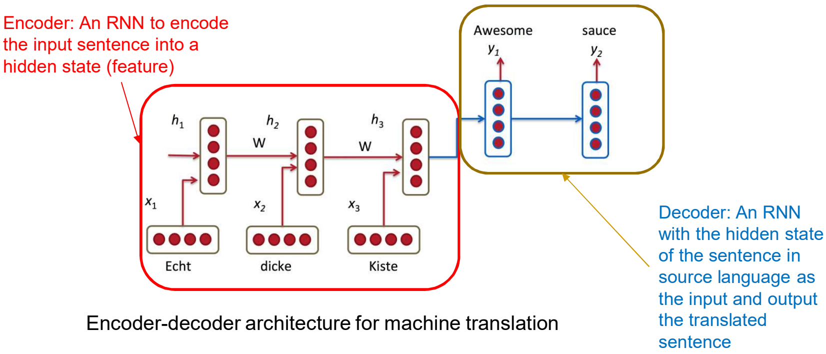
What can RNNs do?

RNNs are generally used for sequence modeling (e.g. language modeling, time series modeling,…).



Machine Translation

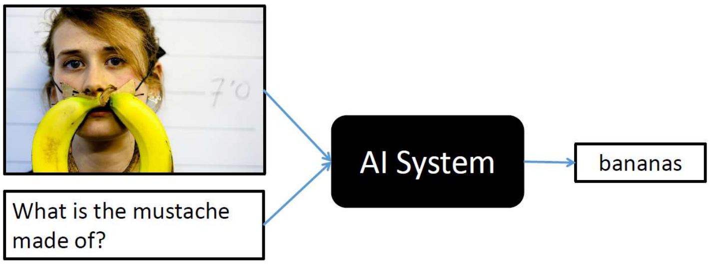
In machine translation, the input is a sequence of words in source language, and the output is a sequence of words in target language.



Encoder-decoder architecture for machine translation

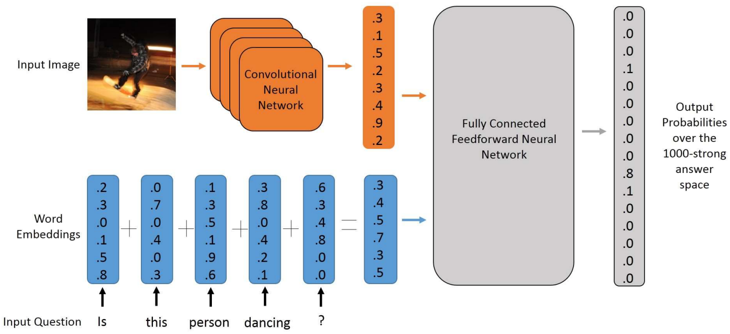
Visual Question Answering (VQA)

VQA: Given an image and a natural language question about the image, the task is to provide an accurate natural language answer



Picture from (Antol et al., 2015)

Visual Question Answering



The output is to be conditioned on both image and textual inputs. A CNN is used to encode the image and a RNN is implemented to encode the sentence.