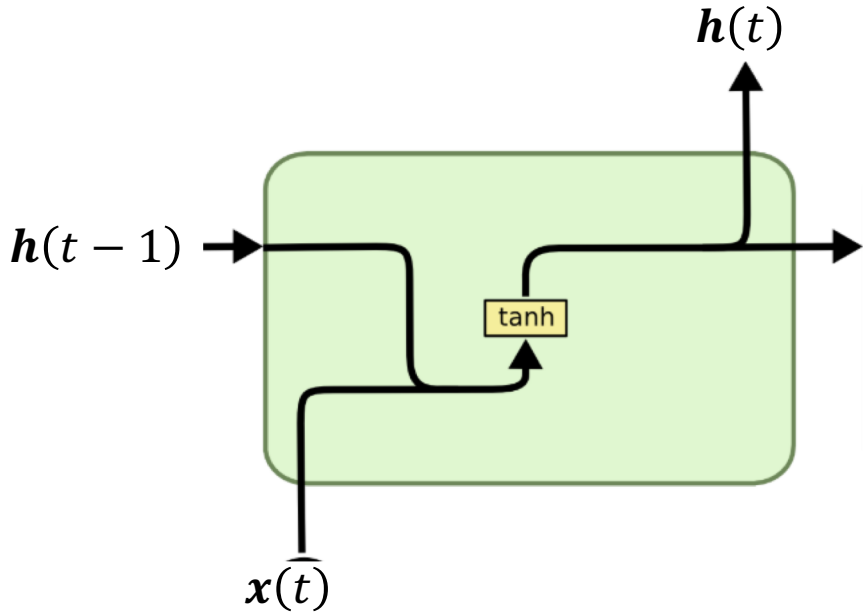
Chapter 9
Gated recurrent neural networks

Neural networks and deep learning

Basic RNN unit



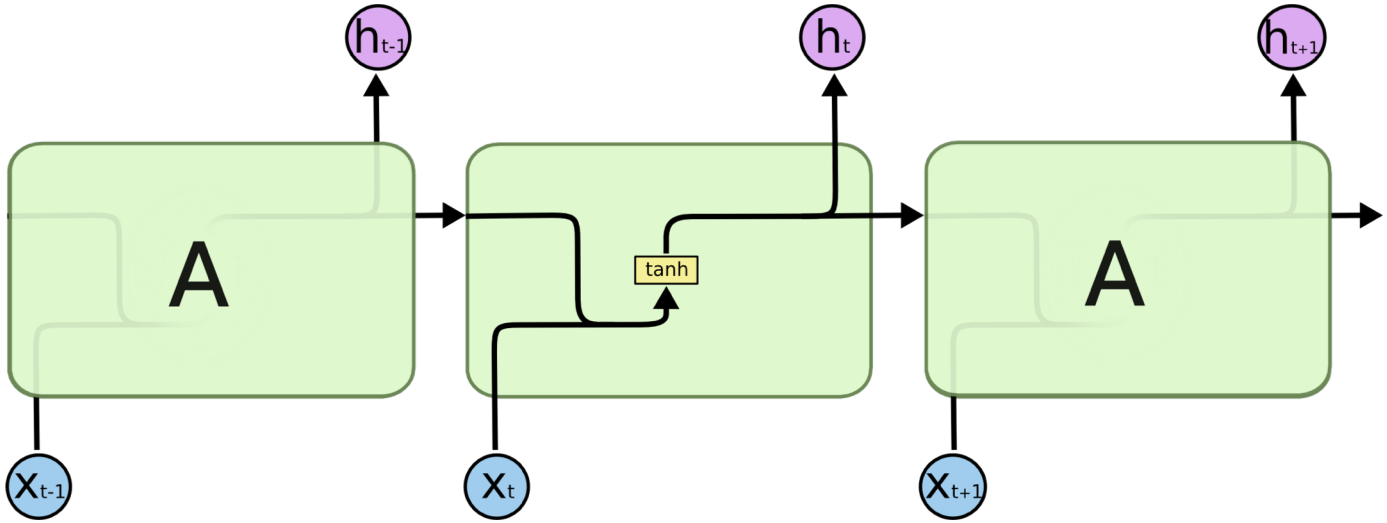
The RNN cell (memory unit) is characterized by

$$\boldsymbol{h}(t) = \phi(\boldsymbol{U}^T\boldsymbol{x}(t) + \boldsymbol{W}^T\boldsymbol{h}(t-1) + \boldsymbol{b})$$

where ϕ is the *tanh* activation function and RNN cell is referred to as Tanh units.

RNN build with simple *tanh* units are also referred to as *vanilla RNN*.

Basic RNN layer



Exploding and vanishing gradients in RNN

Though RNN has been proven to solve sequential problems, it is difficult to train them to learn long term dynamics and dependencies

During gradient back-propagation learning of RNN, the gradient can end up being multiplied a large number of times (as many as the number of time steps) by the weight matrix associated with the connections between the neurons of the recurrent hidden layer.

Note that each time the activations are forward propagated in time, the activations are multiplied by $\boldsymbol{W}$ and each time the gradients are back propagated, the gradients are multiplied by $\boldsymbol{W}^T$.

Exploding and vanishing gradients in RNN

If the weights in this matrix are small. it can lead to a situation called *vanishing gradients* where the gradient signal gets so small that learning either becomes very slow or stops working altogether

Conversely, if the weights in this matrix are large, It can lead to a situation where the gradient signal is so large that it can cause learning to diverge. This is often referred to as *exploding gradients*.

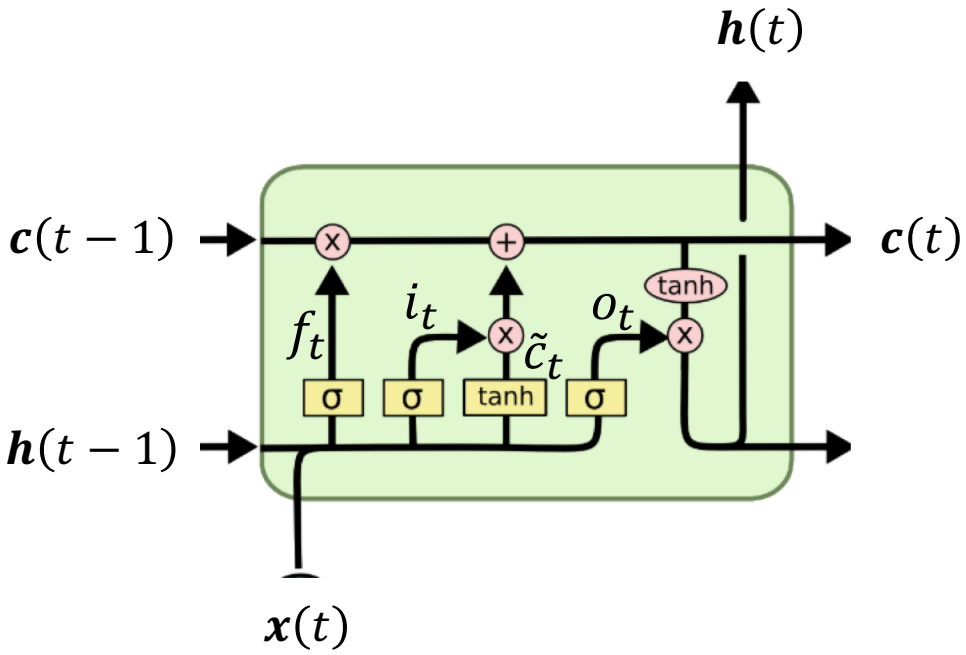
Vanishing and exploding gradients in gradient backpropagation learning makes it difficulty train RNN to learn long-term dependencies.

Exploding and vanishing gradients in RNN

Long-term dependencies are reflections of short-term dependencies in RNN.

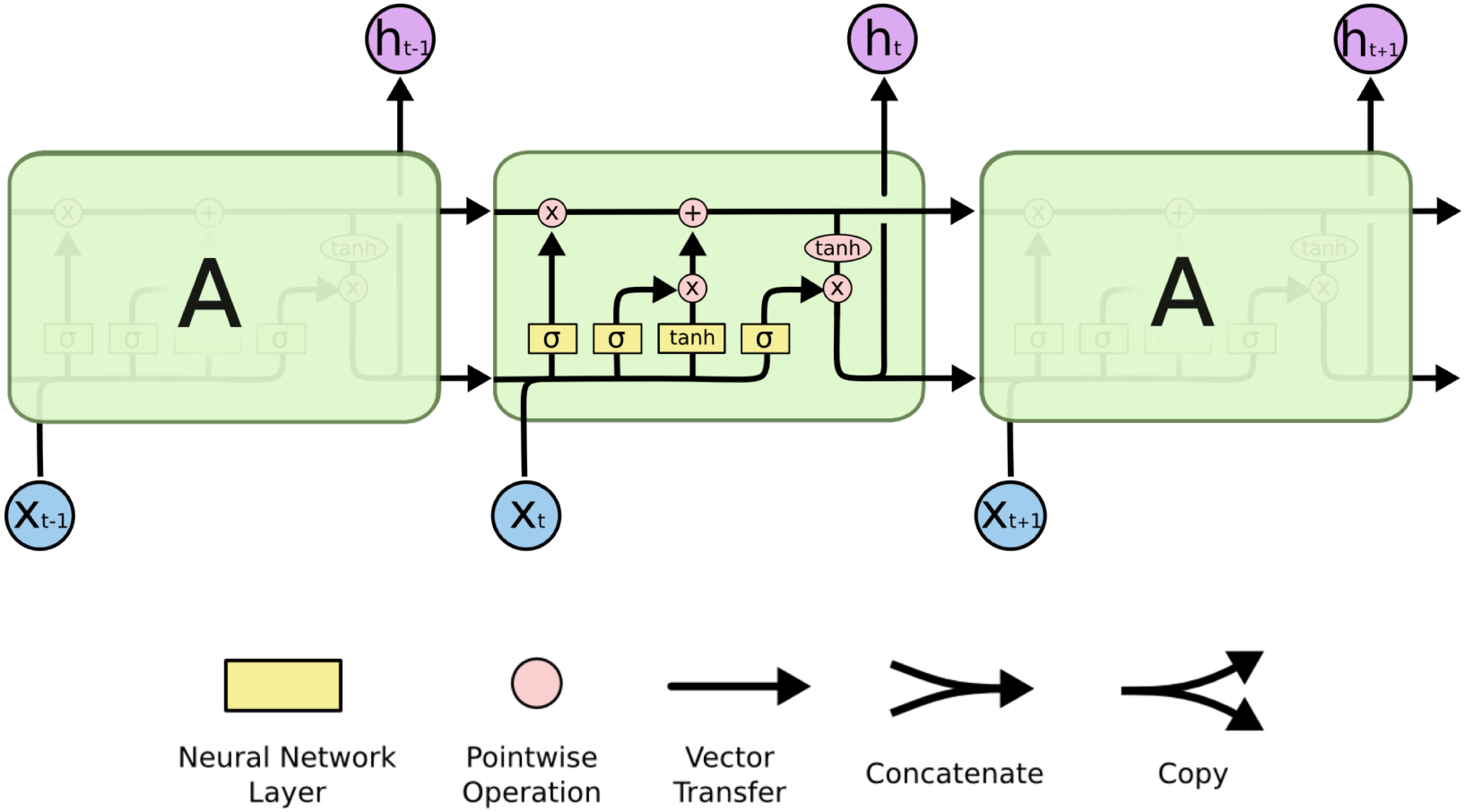
Even if the parameters of recurrent network is stable (can store memories, with gradients not exploding), the difficulty with long-term dependencies arises from the exponentially smaller weights given to long-term interactions (involving the multiplication of many Jacobians) compared to short-term ones.

Long short-term memory (LSTM) unit

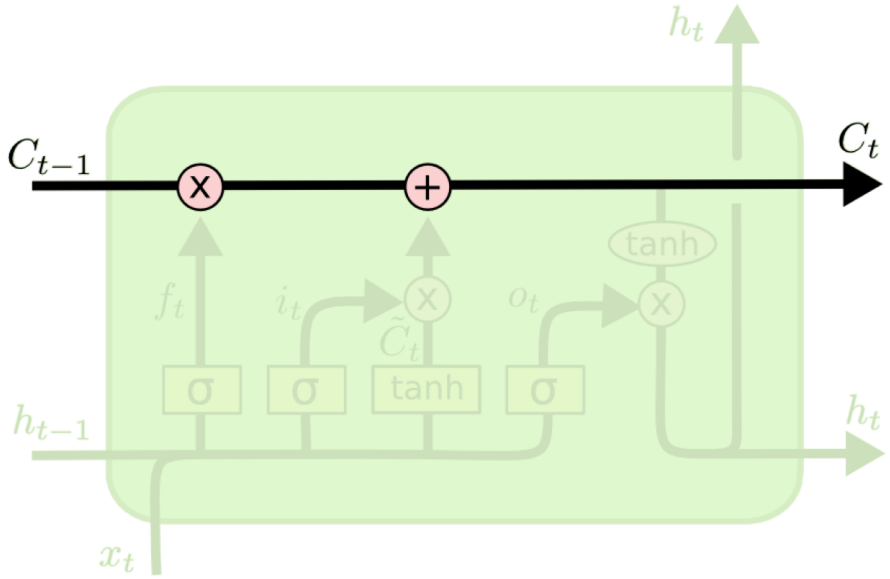


LSTMs provide a solution by incorporating memory units that allow the network to learn when to forget previous hidden states and when to update hidden states given new information.

LSTM layer

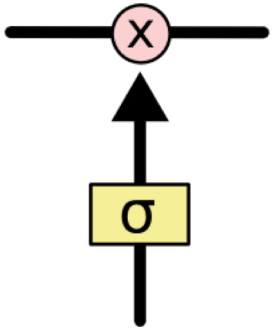

h_{t-1}
h_t
h_{t+1}
A
A
tanh
σ
σ
tanh
σ
X_{t-1}
X_t
X_{t+1}
Neural Network Layer
Pointwise Operation
Vector Transfer
Concatenate
Copy

LSTM unit



The key to LSTM is the cell state $c(t)$ - the horizontal line through the top of the diagram.

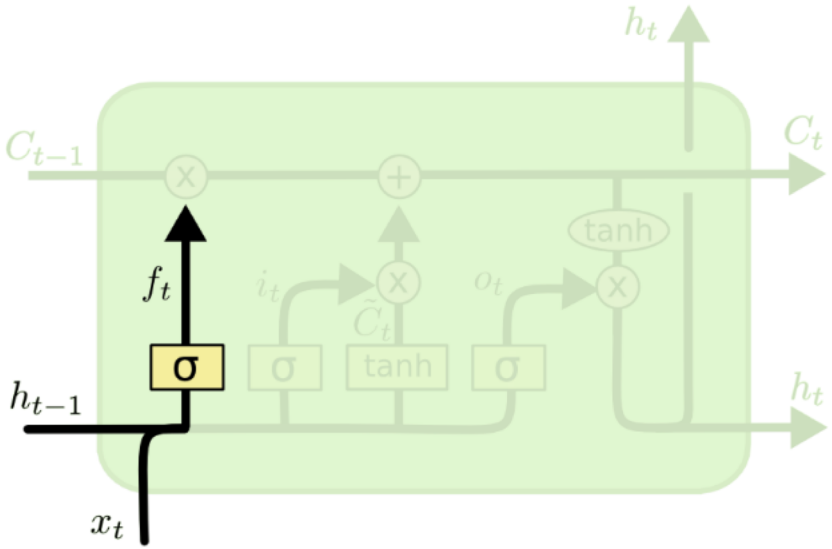
Gates



The LSTM does have the ability to add and remove information to the cell states through gates. Gates are a way to optionally let information through. They are composed of sigmoid neural net layer and pointwise multiplication operations.

The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.

Forget gate

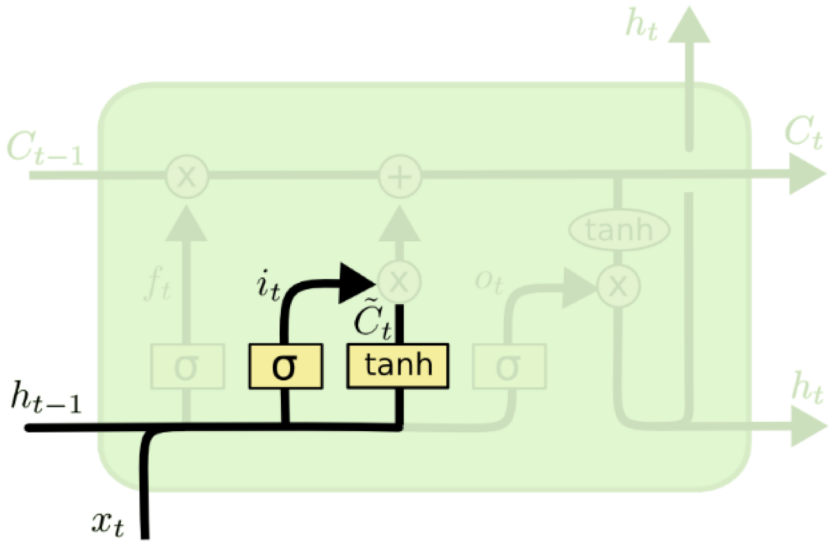


$$\boldsymbol{f}(t) = \sigma\left(\boldsymbol{U_f}^T \boldsymbol{x}(t) + \boldsymbol{W_f}^T \boldsymbol{h}(t-1) + \boldsymbol{b_f}\right)$$

The forget gate can modulate the memory cell's self-recurrent connection, allowing the cell to remember or forget its previous state, as needed.

Value of $\boldsymbol{f}(t)$ determines if $c(t-1)$ is to be remembered or not.

Input gate



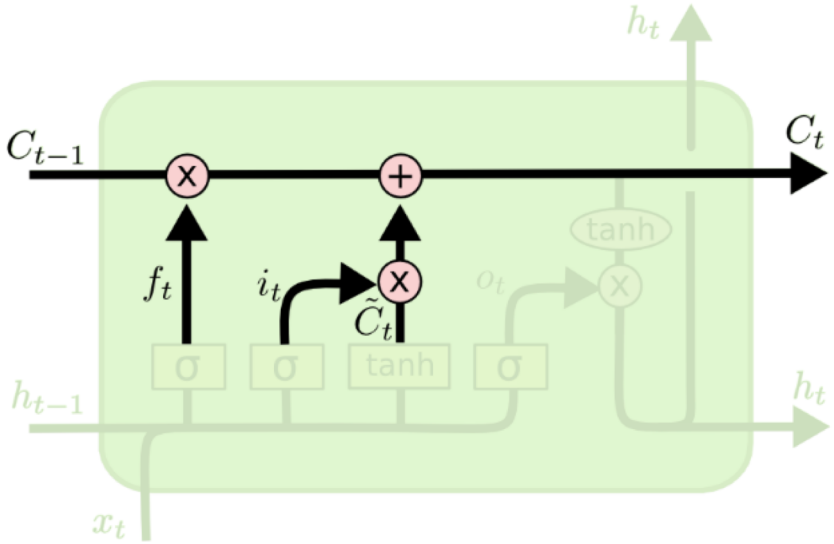
$$\boldsymbol{i}(t) = \sigma\left(\boldsymbol{U}_i{}^T \boldsymbol{x}(t) + \boldsymbol{W}_i{}^T \boldsymbol{h}(t-1) + \boldsymbol{b}_i\right)$$

$$\tilde{\boldsymbol{c}}(t) = \phi\left(\boldsymbol{U}_c{}^T x(t) + \boldsymbol{W}_c{}^T \boldsymbol{h}(t-1) + \boldsymbol{b}_c\right)$$

The input gate can allow incoming signal to alter the state of the memory cell or block it. It decides what new information to store in the cell stage.

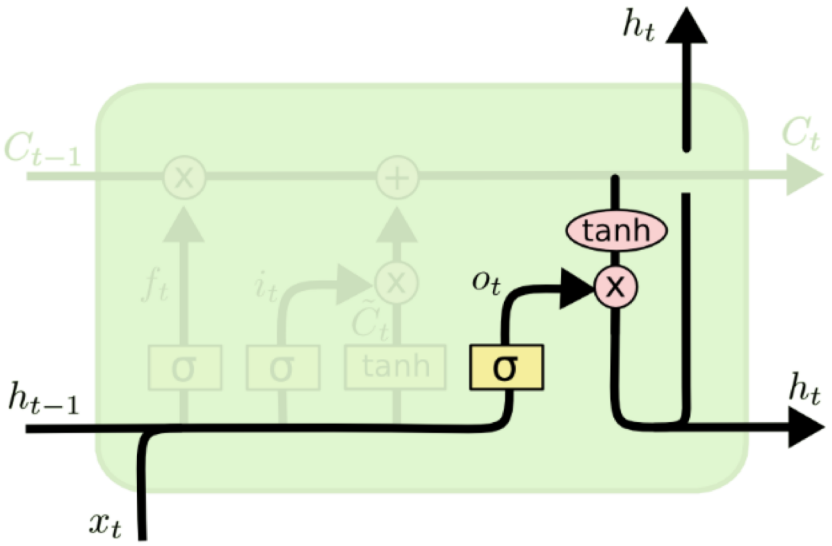
This has two parts: a sigmoid input gate layer decides which values to updates; a tanh layer creates a vector of new candidate values $\tilde{\boldsymbol{c}}(t)$ that could be added to the state.

Cell state



$$\boldsymbol{c}(t) = \tilde{\boldsymbol{c}}(t) \odot \boldsymbol{i}(t) + \boldsymbol{c}(t-1) \odot \boldsymbol{f}(t)$$

Output Gate

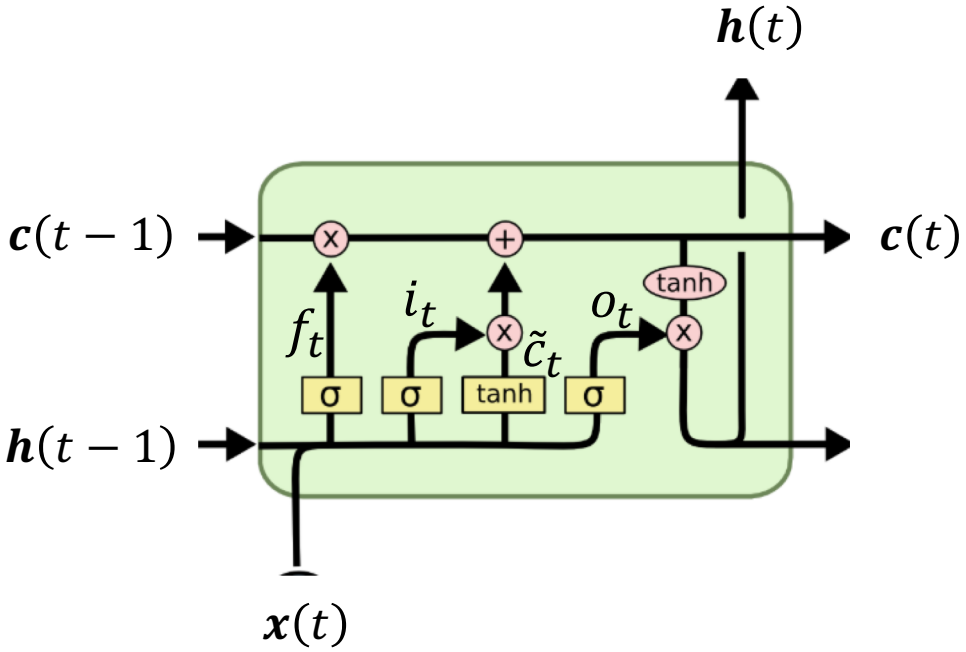


$$\boldsymbol{o}(t) = \sigma\left(\boldsymbol{U}_o{}^T x(t) + \boldsymbol{W}_o{}^T \boldsymbol{h}(t-1) + \boldsymbol{b}_o\right)$$

$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{c}(t)\right) \odot \boldsymbol{o}(t)$$

the output gate can allow the state of the memory cell to have an effect on other neurons or prevent it.

LSTM unit



$$\boldsymbol{i}(t) = \sigma\left(\boldsymbol{U}_i{}^T \boldsymbol{x}(t) + \boldsymbol{W}_i{}^T \boldsymbol{h}(t-1) + \boldsymbol{b}_i\right)$$
$$\boldsymbol{f}(t) = \sigma\left(\boldsymbol{U}_f{}^T \boldsymbol{x}(t) + \boldsymbol{W}_f{}^T \boldsymbol{h}(t-1) + \boldsymbol{b}_f\right)$$
$$\boldsymbol{o}(t) = \sigma\left(\boldsymbol{U}_o{}^T x(t) + \boldsymbol{W}_o{}^T \boldsymbol{h}(t-1) + \boldsymbol{b}_o\right)$$

$$\tilde{\boldsymbol{c}}(t) = \phi\left(\boldsymbol{U}_c{}^T x(t) + \boldsymbol{W}_c{}^T \boldsymbol{h}(t-1) + \boldsymbol{b}_c\right)$$

$$\boldsymbol{c}(t) = \tilde{\boldsymbol{c}}(t) \odot \boldsymbol{i}(t) + \boldsymbol{c}(t-1) \odot \boldsymbol{f}(t)$$
$$\boldsymbol{h}(t) = \phi\left(\boldsymbol{c}(t)\right) \odot \boldsymbol{o}(t)$$

LSTM units

LSTM introduces a gated cell where the information flow can be controlled. ‘’cells’’ in LSTM have an internal recurrence (a self-loop) in addition to the outer recurrence of the RNN.

The most important component is the state unit $c_i(t)$which has a linear self-loop. The self-loop weight is controlled by a forget gate unit $c_i(t)$ (for time step t and cell i), which sets this weight to a value between 0 and 1 via a sigmoid unit.

Each cell has the same inputs and outputs as an ordinary recurrent network, but also has more parameters and a system of gating units that control the flow of information.

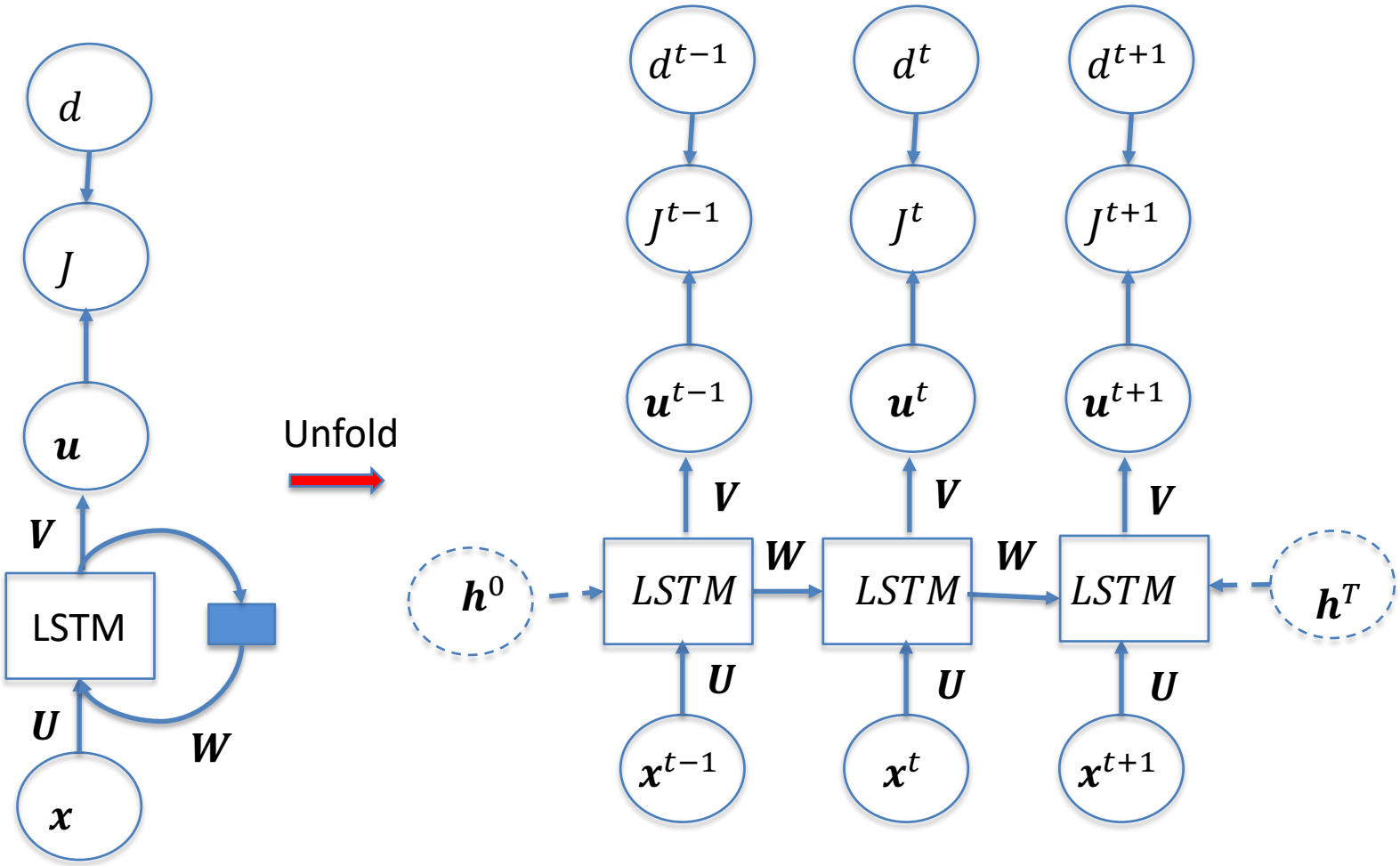
LSTM unit

The self-loop inside the cell is able to produces paths where the gradient can flow for long duration and to make the weight on this self-loop conditioned on the context rather than fixed.

By making the weight of this self-loop gated (controlled by another hidden unit), the time scale of integration can be changed dynamically. The time scale of integration is based on the input sequence, because time constants are output by the model itself.

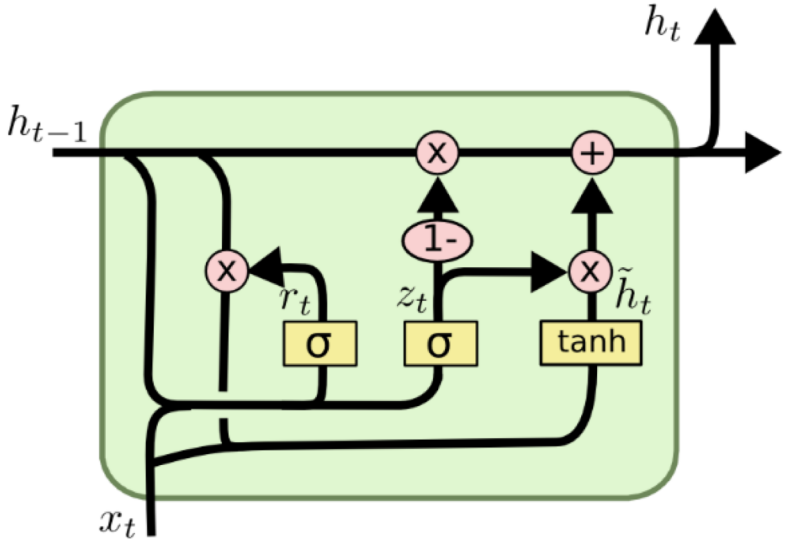
In this way, LSTM help preserve error terms that can be propagated through many layers and time steps.

Training LSTM networks



$\boldsymbol{W} = \{\boldsymbol{W_i}, \boldsymbol{W_f}, \boldsymbol{W_o}, \boldsymbol{W_c}\}$, $\boldsymbol{U} = \{\boldsymbol{U_i}, \boldsymbol{U_f}, \boldsymbol{U_o}, \boldsymbol{U_c}\}$, and $\boldsymbol{b} = \{\boldsymbol{b_i}, \boldsymbol{b_f}, \boldsymbol{b_o}, \boldsymbol{b_c}\}$

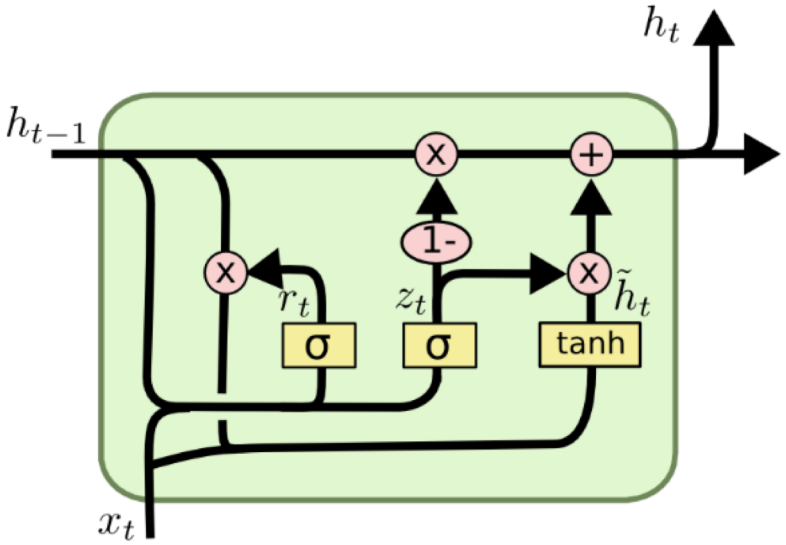
Gated Recurrent Unit (GRU)



Like LSTM units, *gated recurrent units* (GRU) attempt to create paths through time that have gradients either vanish or explode.

The main difference with the LSTM is that a single gating unit simultaneously controls the forgetting factor and the decision to update the hidden unit.

GRU

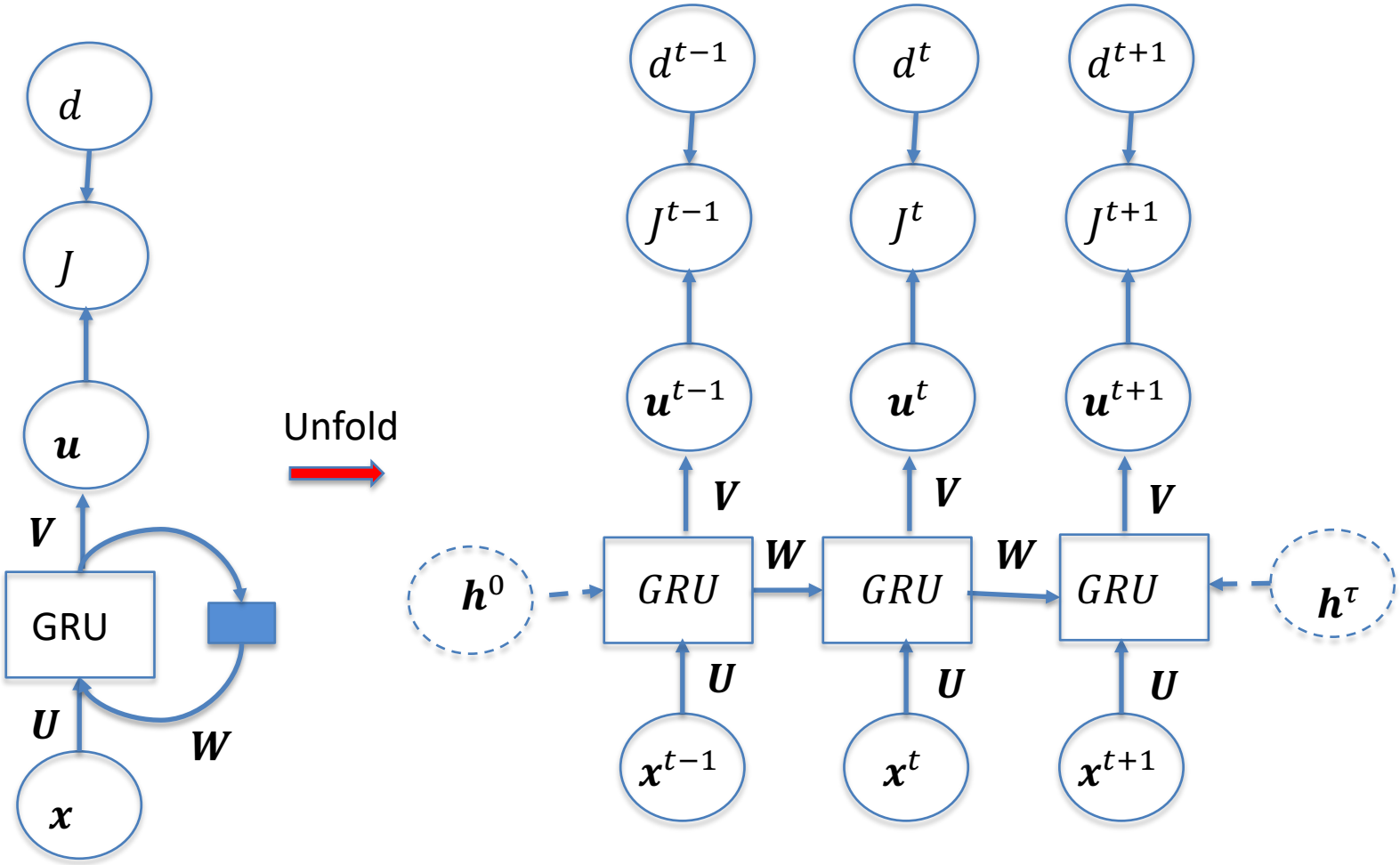


$$\boldsymbol{r}(t) = \sigma\left(\boldsymbol{U}_r{}^T\boldsymbol{x}(t) + \boldsymbol{W}_r{}^T\boldsymbol{h}(t-1) + \boldsymbol{b}_r\right)$$
$$\boldsymbol{z}(t) = \sigma\left(\boldsymbol{U}_z{}^T\boldsymbol{x}(t) + \boldsymbol{W}_z{}^T\boldsymbol{h}(t-1) + \boldsymbol{b}_z\right)$$

$$\widetilde{\boldsymbol{h}}(t) = \phi\left(\boldsymbol{U}_h{}^T\boldsymbol{x}(t) + \boldsymbol{W}_h{}^T\left(\boldsymbol{r}(t)\odot\boldsymbol{h}(t-1)\right) + \boldsymbol{b}_h\right)$$
$$\boldsymbol{h}(t) = \left(1 - \boldsymbol{z}(t)\right)\odot\boldsymbol{h}(t-1) + \boldsymbol{z}(t)\odot\widetilde{\boldsymbol{h}}(t)$$

$\boldsymbol{r}$ is the *reset gate* and $\boldsymbol{z}$ *update gate* inputs.
$\widetilde{\boldsymbol{h}}(t)$ is the candidate hidden activation.

Training GRU networks



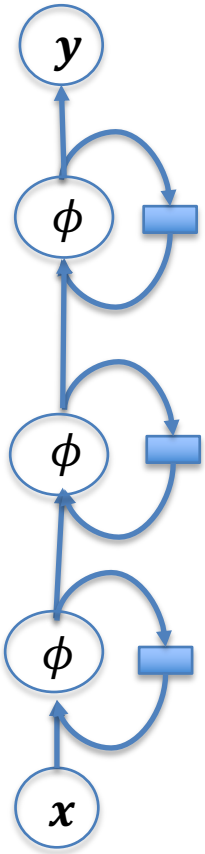
$\boldsymbol{W} = \{\boldsymbol{W_r}, \boldsymbol{W_z}, \boldsymbol{W_h}\}$, $\boldsymbol{U} = \{\boldsymbol{U_r}, \boldsymbol{U_z}, \boldsymbol{U_h}\}$, and $\boldsymbol{b} = \{\boldsymbol{b_r}, \boldsymbol{b_z}, \boldsymbol{b_h}\}$

LSTM and GRU

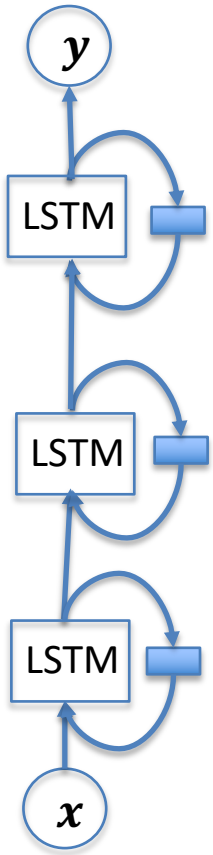
The update gate determines how much of the past information from previous time steps is passed through to the future. The reset gate determines how much of the information to forget. There is no persistent cell state in GRU as in the case of LSTM.

Unlike vanilla-RNN which overwrites its content at each time-step, GRU and LSTM unit is able to decide whether to keep the existing memory via the introduced gates. Intuitively, if the LSTM unit detects an important feature from an input sequence at early stage,, it easily carries this information over a long distance, hence capturing potential long distance dependencies. Both GRU and LSTM overcomes the issues with exploding and vanishing gradients, and perform well in complex analysis of sequence information.

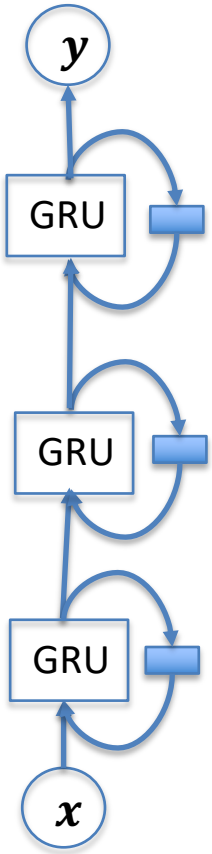
Deep LSTM and Deep GRUs



RNN



LSTM networks



GRNN

Example 1

Generate 64 2-dimensional input sequences $\left(\boldsymbol{x}(t)\right)_{t=1}^{16}$ where $\left(x_1(t), x_2(t)\right) \in [0,1]^2$ by randomly generating numbers uniformly.

Generate 1-dimensional output sequences $\left(y(t)\right)_{t=1}^{16}$ where $y(t) \in \boldsymbol{R}$ by following the following recurrent relation:

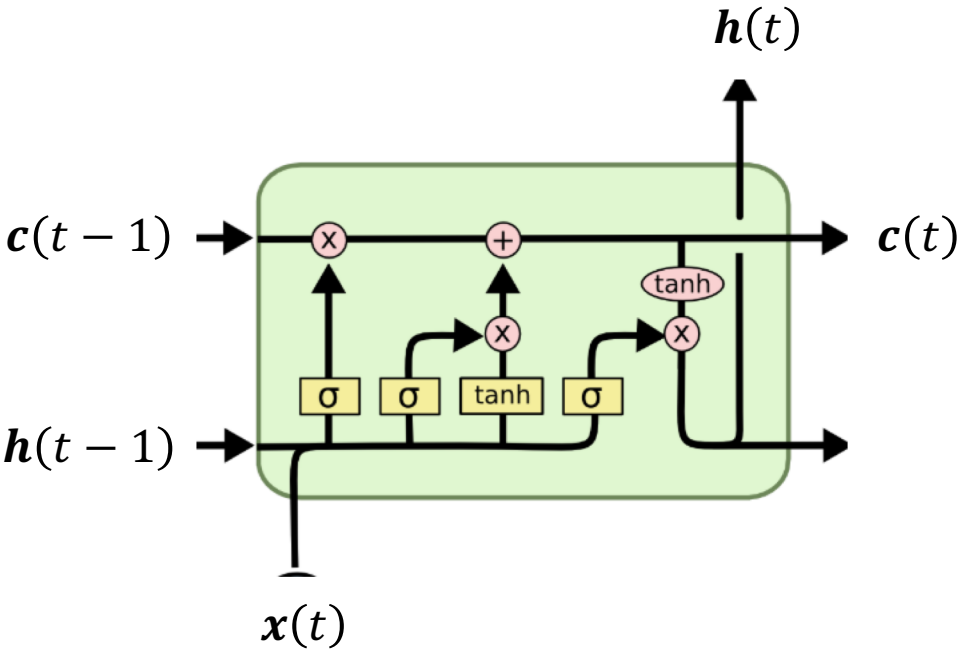
$$y(t) = 5x_1(t-1)x_2(t-1) - 2x_1(t-7) + 3.5{x_2}^2(t-5) + 0.1\epsilon$$

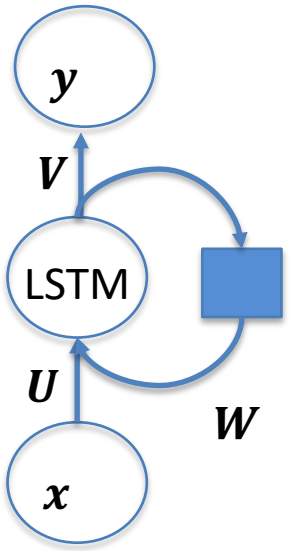
where $\epsilon \sim N(0,1)$.

Train a LSTM layer to learn the mapping between input and output sequences.

Use a learning factor $\alpha = 0.001$.

Repeat the above using RNN and GRU and compare the performances.





$W = \{W_i, W_f, W_o, W_c\}$, $U = \{U_i, U_f, U_o, U_c\}$, and $b = \{b_i, b_f, b_o, b_c\}$

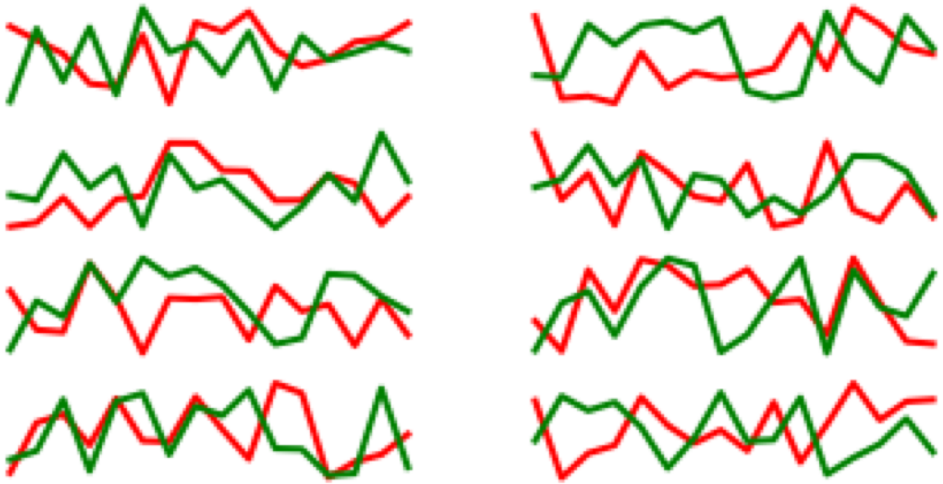
n_in = 2
n_hidden = 10
n_out = 1
n_steps = 16
n_seqs = 64

$$y(t) = 5x_1(t-1)x_2(t-1) - 2x_1(t-7) + 3.5{x_2}^2(t-5)$$

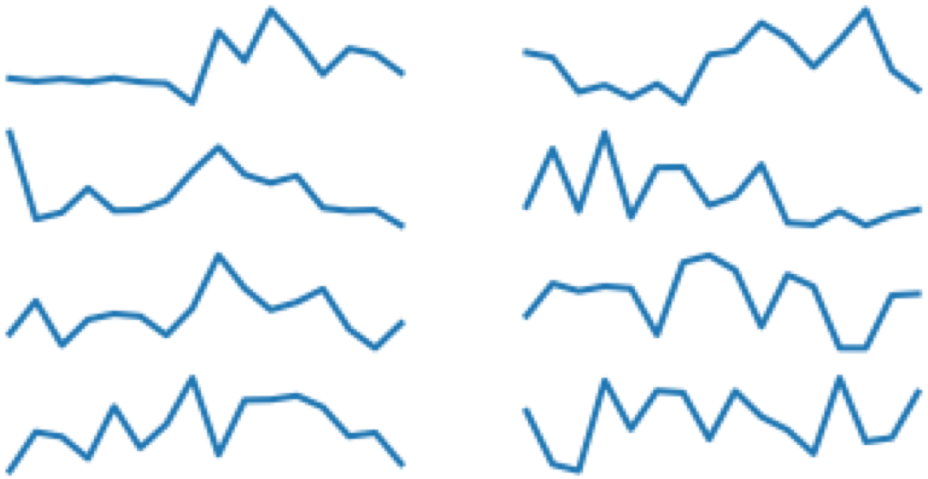
x_train = **np.random.rand**(n_seqs*n_steps, n_in)

y_train = **np.zeros**([n_seqs*n_steps, n_out])
y_train[7:,0] = 5*x_train[6:-1, 0]*x_train[5:-2,1] - 2*x_train[:-7,0] + 3.5*x_train[2:-5,1]**2
y_train += 0.1***np.random.randn**(n_seqs*n_steps, n_out)

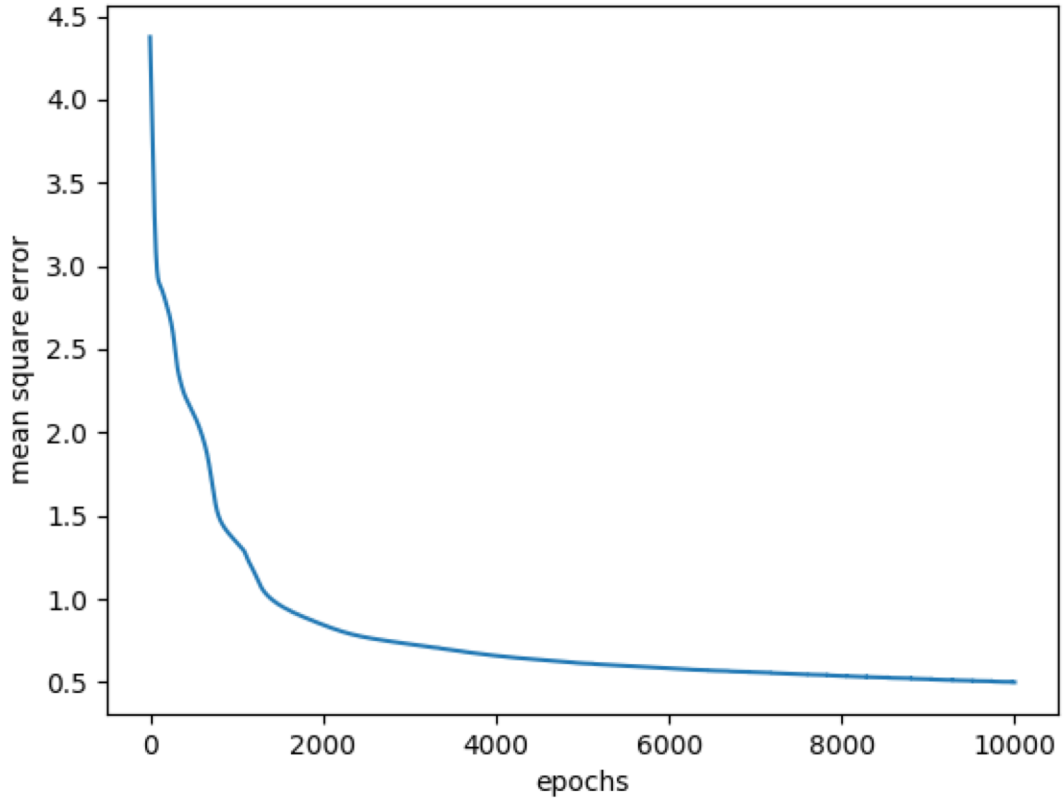
$x_1(t)$ and $x_2(t)$



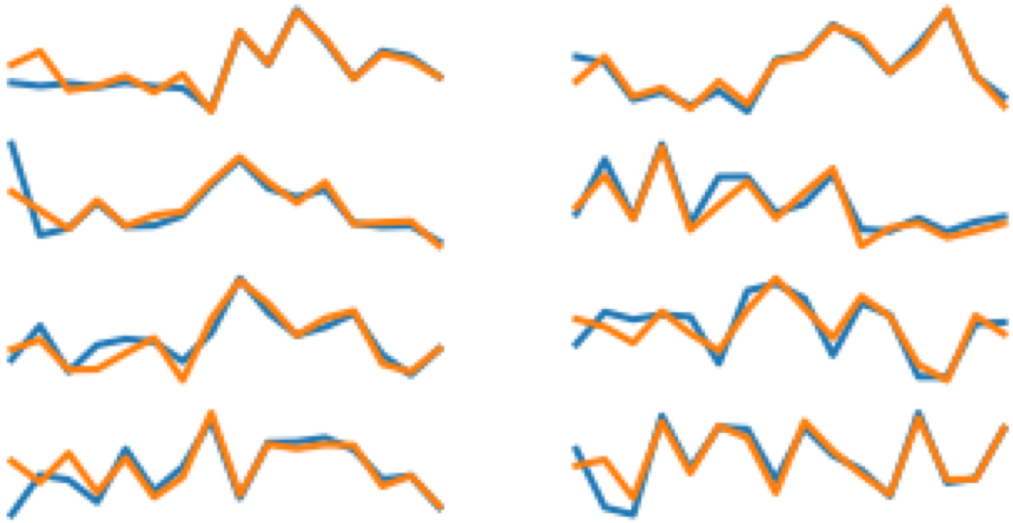
$$y(t) = 5x_1(t-1)x_2(t-1) - 2x_1(t-7) + 3.5{x_2}^2(t-5)$$

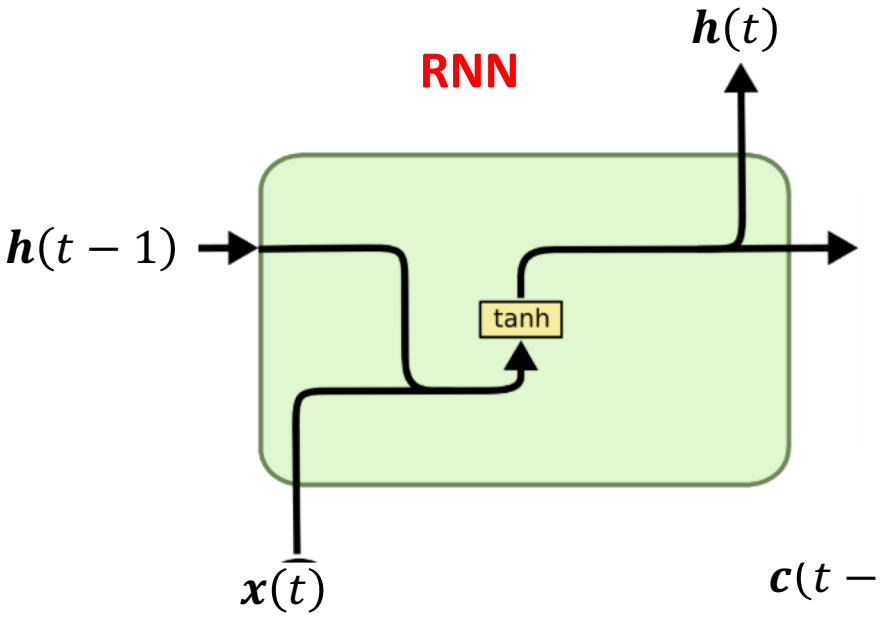


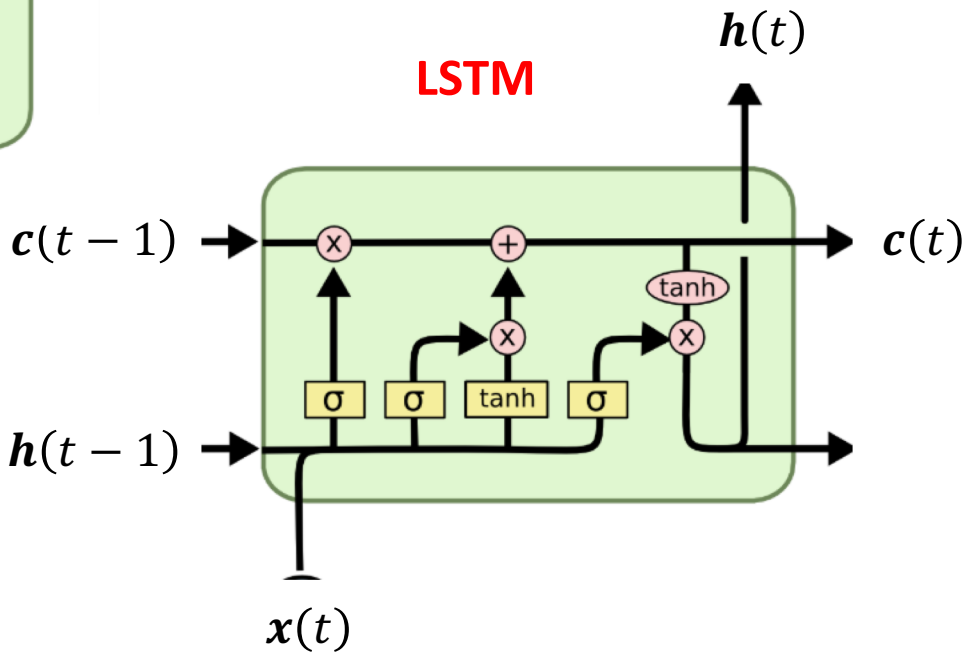
```
cell = tf.nn.rnn_cell.BasicLSTMCell(n_hidden)
outputs, states = tf.nn.dynamic_rnn(cell, x, dtype=tf.float32)

ys = []
for i, h in enumerate(tf.split(outputs, n_steps, axis = 1)):
    y_ = tf.matmul(tf.squeeze(h), W) + b
    ys.append(y_)

ys_ = tf.stack(ys, axis=1)
cost = tf.reduce_mean(tf.reduce_sum(tf.square(y - ys_), axis=2))
train_op = tf.train.AdamOptimizer(lr).minimize(loss=cost)
```

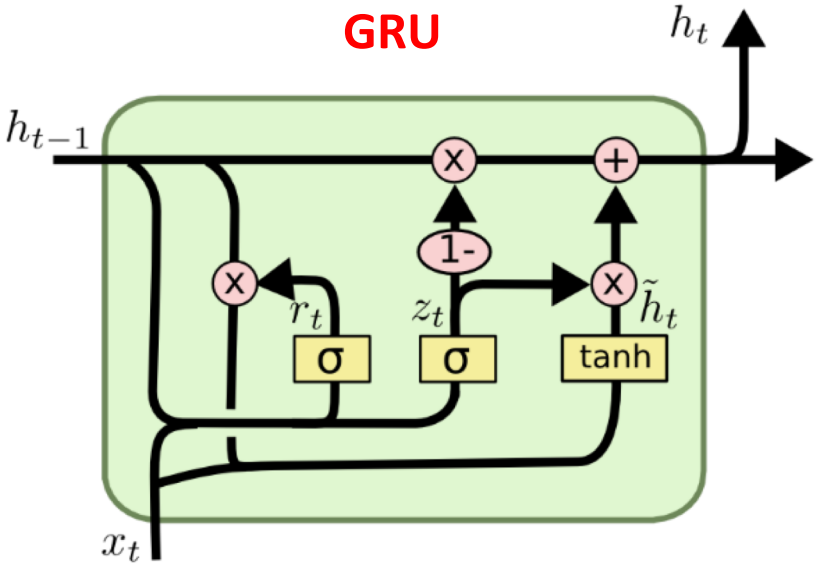

4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
mean square error
0
2000
4000
6000
8000
10000
epochs

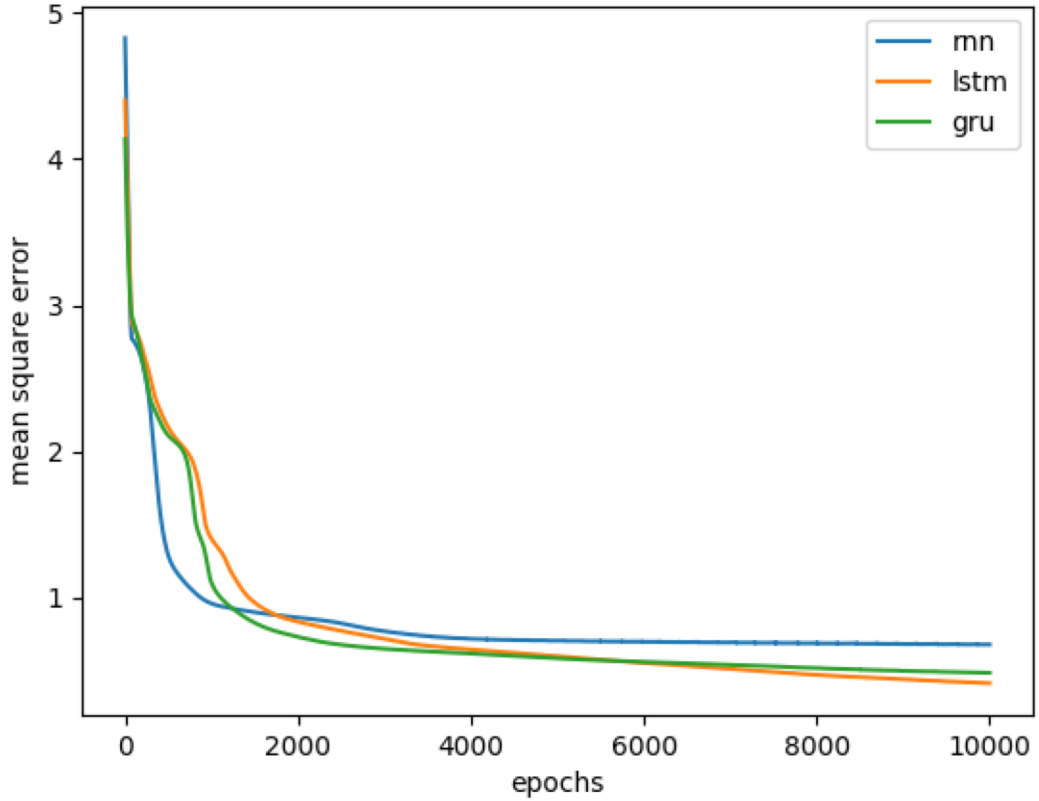
Trained and predicted time-series




RNN
$\boldsymbol{h}(t)$
$\boldsymbol{h}(t-1)$
tanh
$\boldsymbol{x}(t)$


LSTM
$\boldsymbol{h}(t)$
$\boldsymbol{c}(t-1)$
$\boldsymbol{c}(t)$
tanh
σ
σ
tanh
σ
$\boldsymbol{h}(t-1)$
$\boldsymbol{x}(t)$


GRU
h_t
h_{t-1}
1-
r_t
z_t
$\tilde{h}_t$
σ
σ
tanh
x_t


mn
lstm
gru
mean square error
epochs
5
4
3
2
1
0
2000
4000
6000
8000
10000

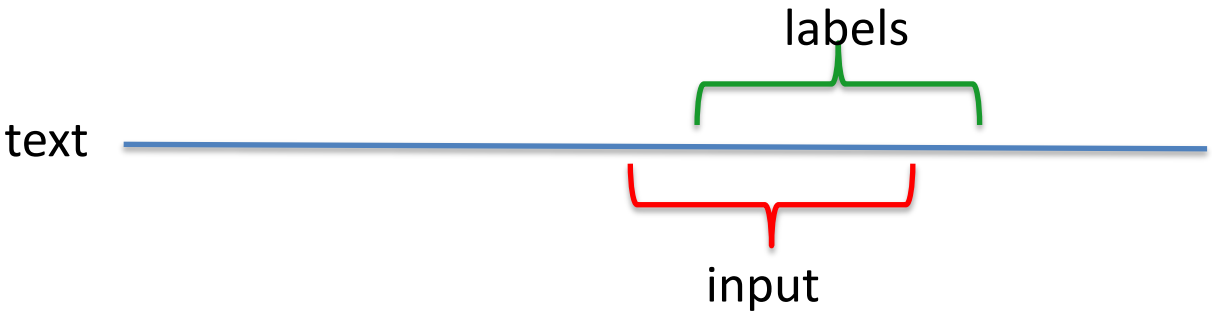
Language Modeling

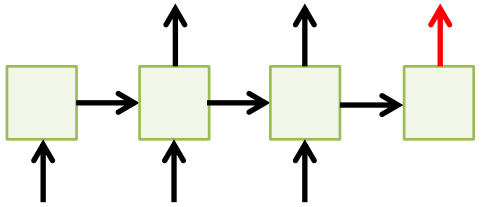
VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare 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.

Text can be modelled at the character-level or word-level
The text will be replaced by either character or word IDs

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

Language modeling





For example, the RNN attempts to predict the next character.

Example 2

Generate 256 points of a time-series $\left(x(t)\right)_{t=1}^{256}$ with a recurrent relation:

$$x(t) = 0.4x(t-1)x(t-3) + 0.2x(t-5) + 0.3x(t-7) + 0.1\epsilon$$

where $\epsilon = N(0,1)$.

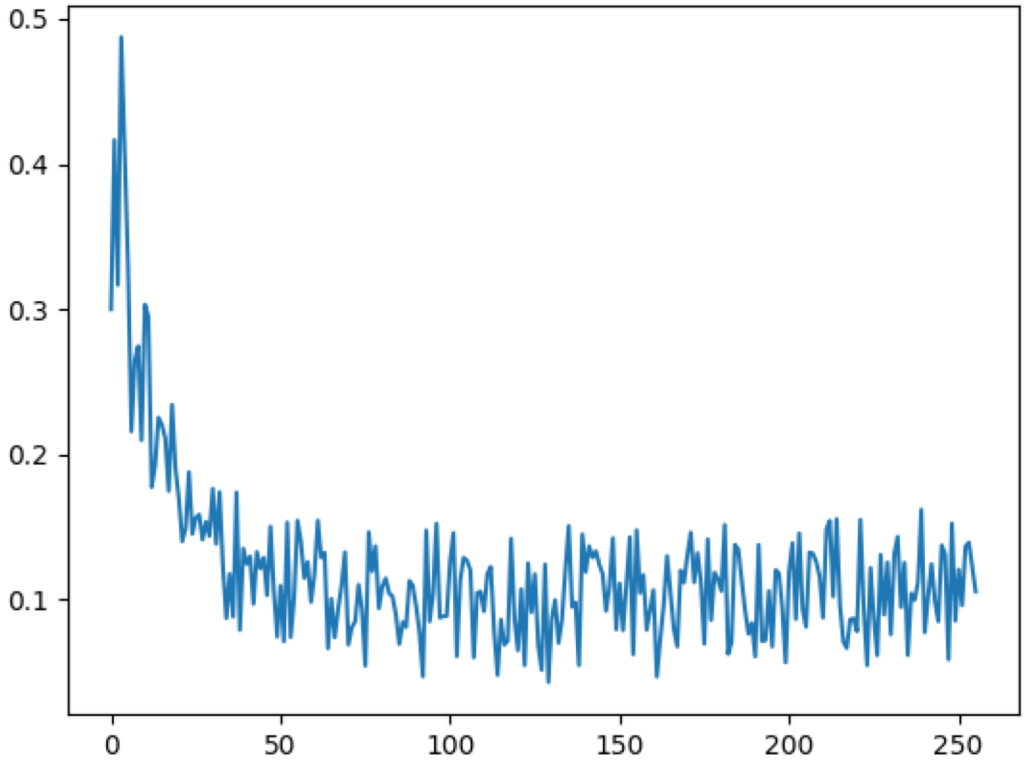
Train an LSTM layer to predict the values of the time-series by using 8 previous time points.

Compare the performance with a 2-layer LSTM networks.

$$x(t) = 0.4x(t-1)x(t-3) + 0.2x(t-5) + 0.3x(t-7) + 0.1\epsilon$$

```
seq = []
for i in range(n_steps):
    seq.append(random.uniform(0, 1))
for i in range(sequence_length):
    seq.append(0.4*seq[-1]*seq[-3] + 0.2*seq[-5] + 0.3*seq[-7] + 0.1*random.random())
```

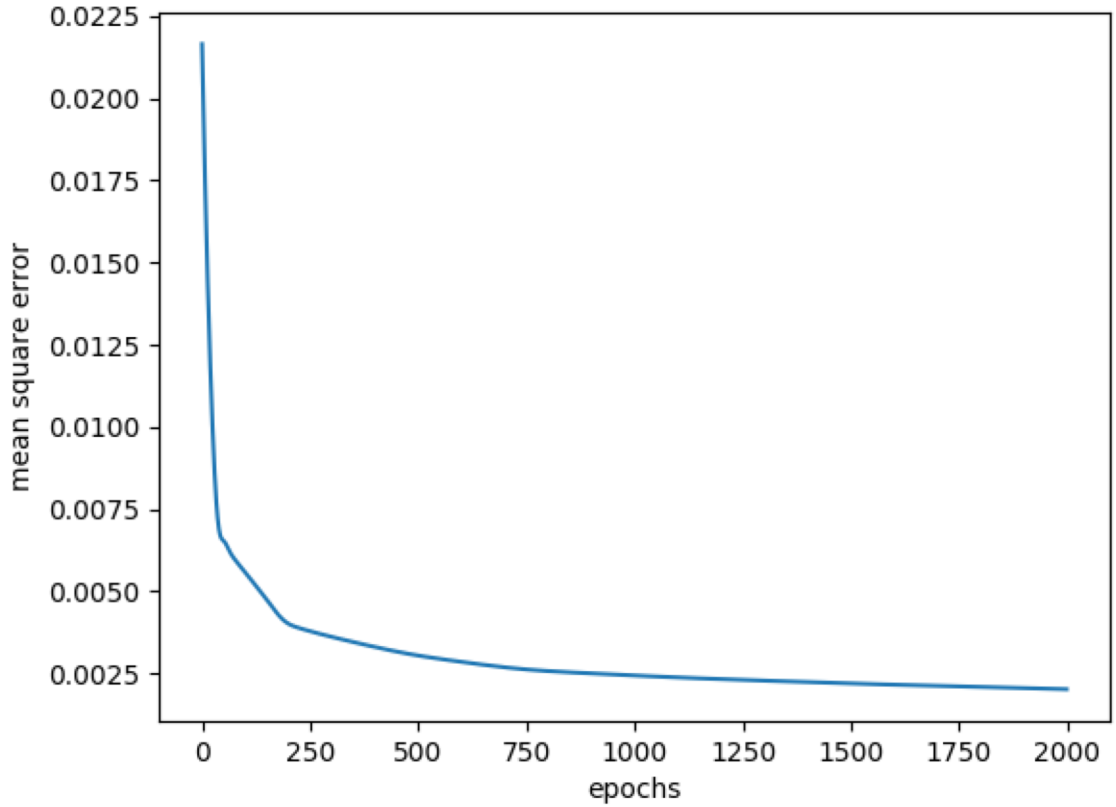
$$x(t) = 0.4x(t-1)x(t-3) + 0.2x(t-5) + 0.3x(t-7) + 0.1\epsilon$$



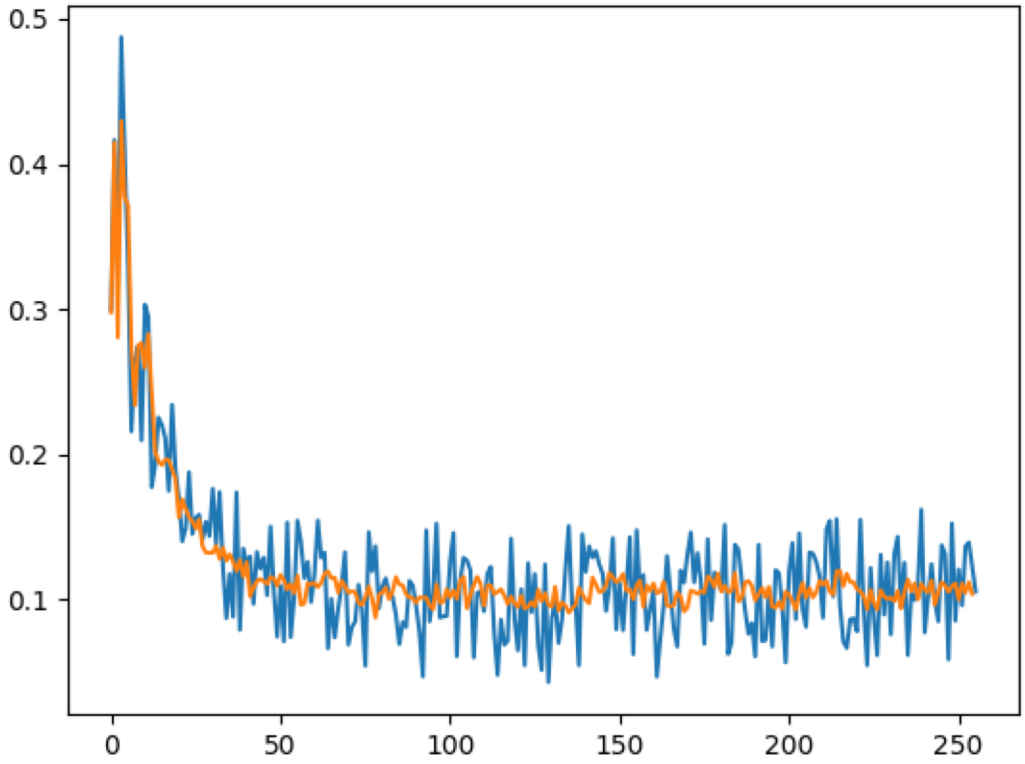
```
x_train, y_train = [], []
for i in range(len(seq) - n_steps - 1):
    x_train.append(np.expand_dims(seq[i:i+n_steps], axis=1).tolist())
    y_train.append(np.expand_dims(seq[i+1:i+n_steps+1], axis=1).tolist())
```

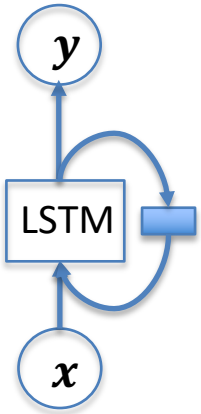
```
cell1 = tf.nn.rnn_cell.BasicLSTMCell(n_hidden, reuse=tf.get_variable_scope().reuse)
cell2 = tf.nn.rnn_cell.BasicLSTMCell(n_hidden, reuse=tf.get_variable_scope().reuse)
cells = tf.nn.rnn_cell.MultiRNNCell([cell1, cell2])
outputs, states = tf.nn.dynamic_rnn(cells, x, dtype=tf.float32)

ys = []
for i, h in enumerate(tf.split(outputs, n_steps, axis = 1)):
    y_ = tf.matmul(tf.squeeze(h, [1]), W) + b
    ys.append(y_)

ys_ = tf.stack(ys, axis=1)
cost = tf.reduce_mean(tf.reduce_sum(tf.square(y - ys_), axis=2))
train_op = tf.train.AdamOptimizer(lr).minimize(loss=cost)
```

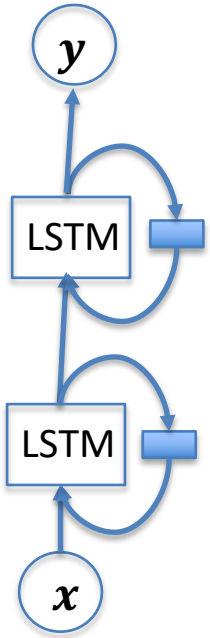
Single-layer LSTM

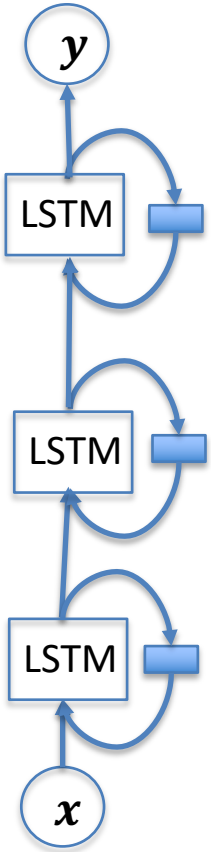


Single-layer LSTM

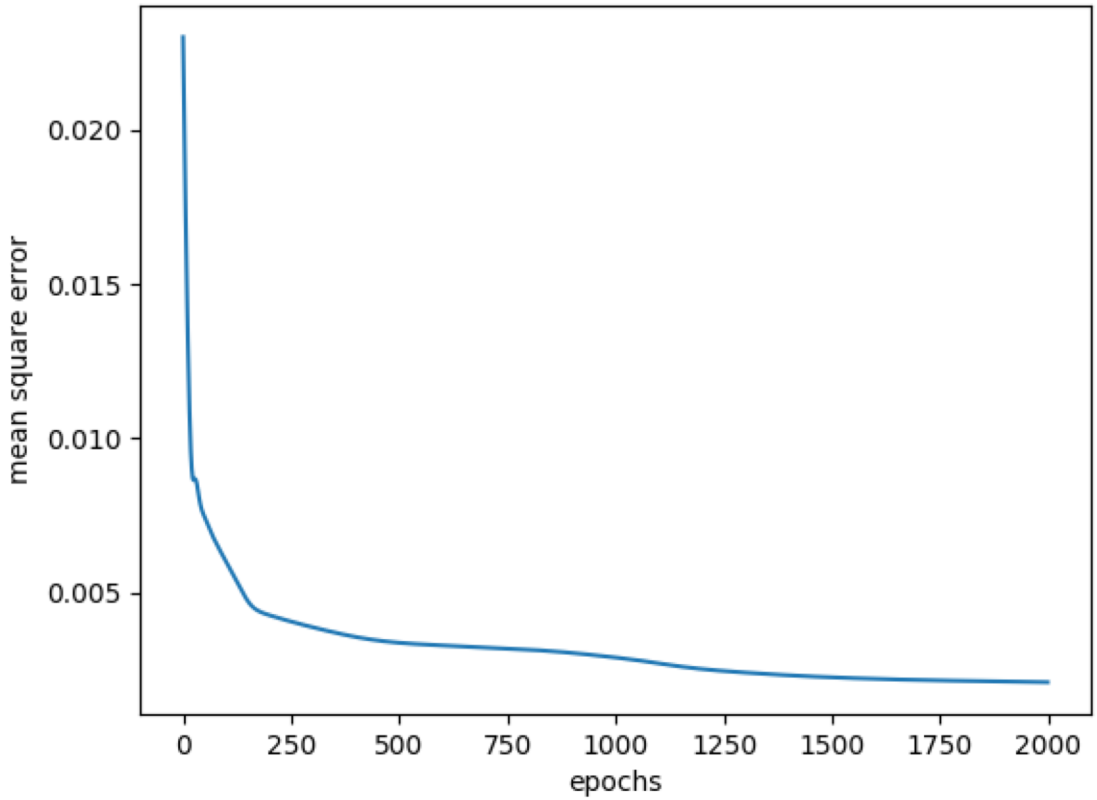



y
LSTM
x

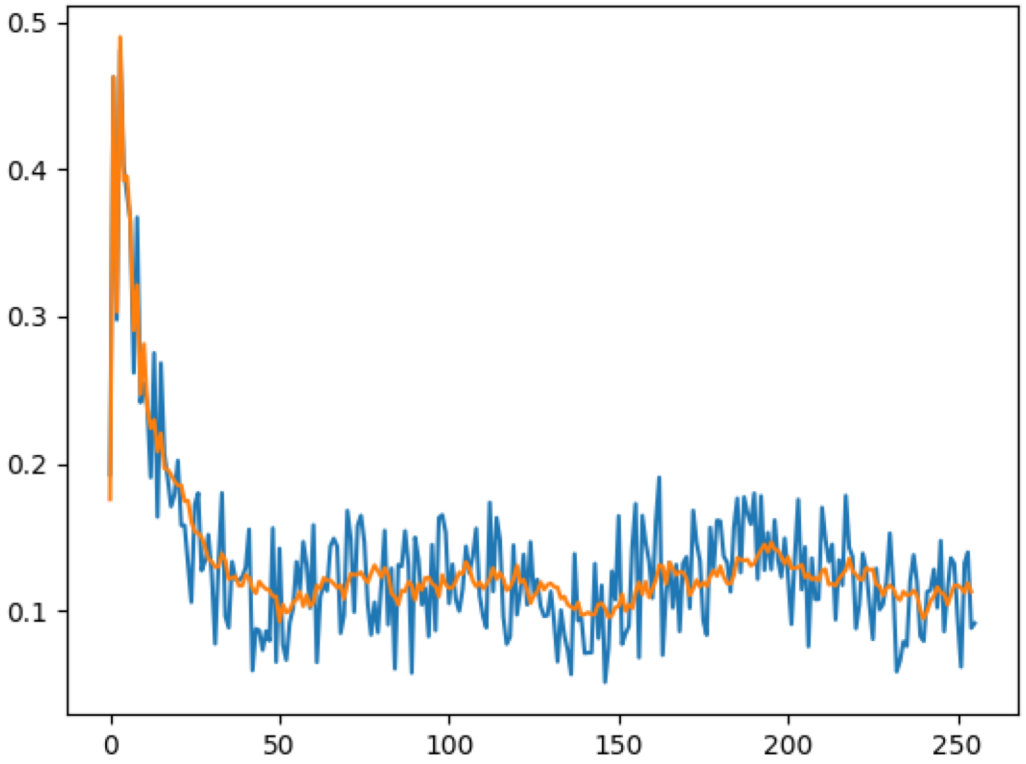

y
LSTM
LSTM
x


y
LSTM
LSTM
LSTM
x

Two-layer LSTM



Two-layer LSTM



Text classification: Dbpedia dataset

This dataset contains first paragraph of Wikipedia page of 56000 entities and label them as one of 15 categories like people, company, schools, etc.

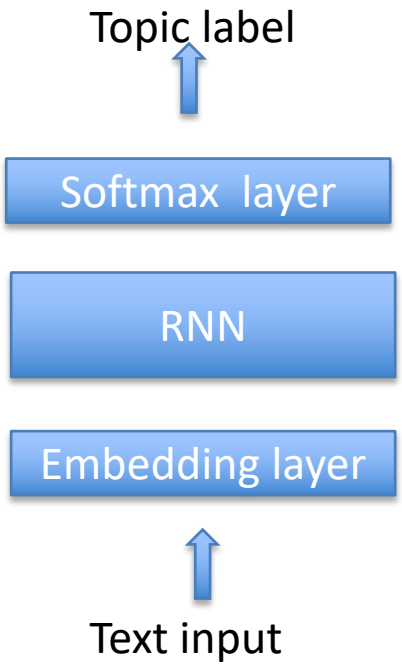
1	D. C. Thomson & Co.	D. C. Thomson & Company Limited is a Scottish publishing company based in Dundee best known for
1	Pyro Spectaculars	Pyro Spectaculars is headquartered in Rialto California USA and occupies a portion of a former World V
1	Admiral Insurance	Admiral a trading name of EUI Limited is a car insurance specialist which launched in January 1993. Its
1	Pax Softnica	Pax Softnica (パックスソフトニカ) is a Japanese video game developer founded in 1983 under the nam
1	The ACME Laboratories Ltd	The ACME Laboratories Ltd is a major pharmaceutical company based in Bangladesh. It is part of the A
2	Dubai Gem Private School & Nursery	Dubai Gem Private School (DGPS) is a British school located in the Oud Metha area of Dubai United Ara
2	Michael Wallace Elementary School	Michael Wallace Elementary School is a Canadian public school in Dartmouth Nova Scotia. It is operate
2	Alma Heights Christian Schools	Alma Heights Christian Schools (AHC) formerly Alma Heights Christian Academy is a private elementar

An input is a text
Requires to find all words in the text and remap them into word IDs, a number per each unit word (word-level inference)

my work is cool! → [23, 500, 5, 1402, 17]

We need to make sure that each sentence is of same length (no_words). The maximum document length is fixed and longer sentences will be truncated and shorter once are padded with zeros.

Text classification



The embedding layer converts one-hot vector of word representations (of size of the vocabulary) to a vector of fixed length (embedding_size) vectors.

Embedding layer learns a weight matrix of [vocab_size, embedding_size]
And then maps word indexes of the sequences into
[batch_size, sequence_length, embedding_size] input to the RNN.

Sentiment Classification

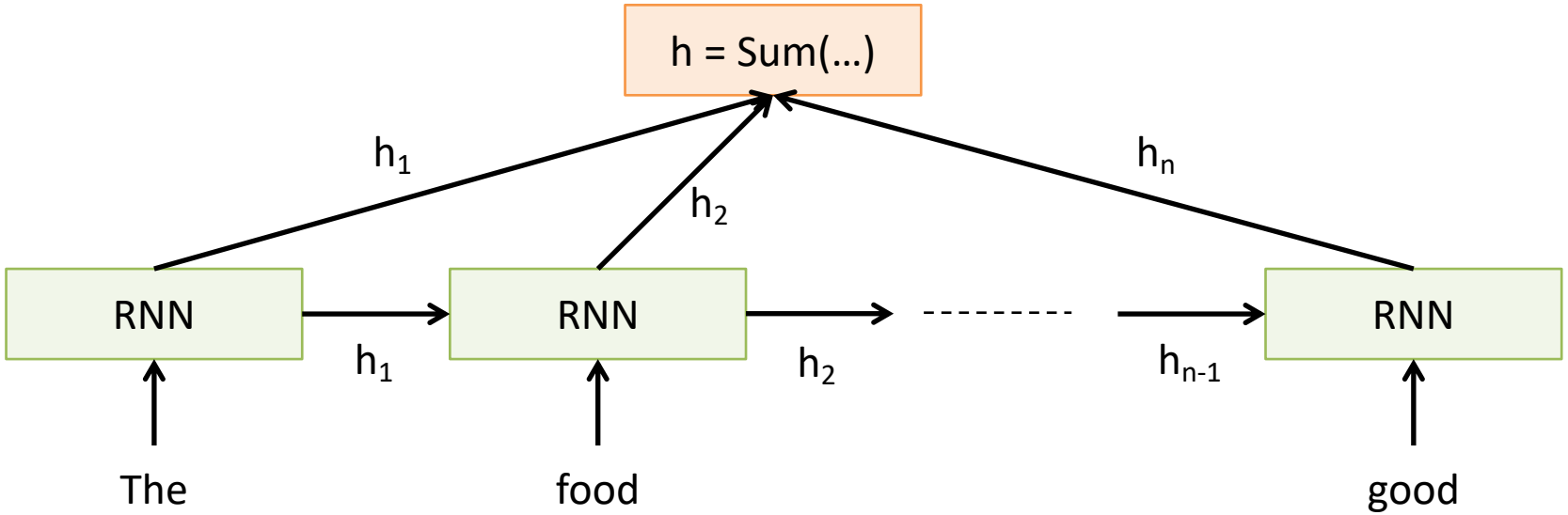
- “The food was really good”

“The chicken crossed the road because it was uncooked”

- Classify a
  restaurant review from Yelp! OR
  movie review from IMDB OR
  …
  as positive or negative

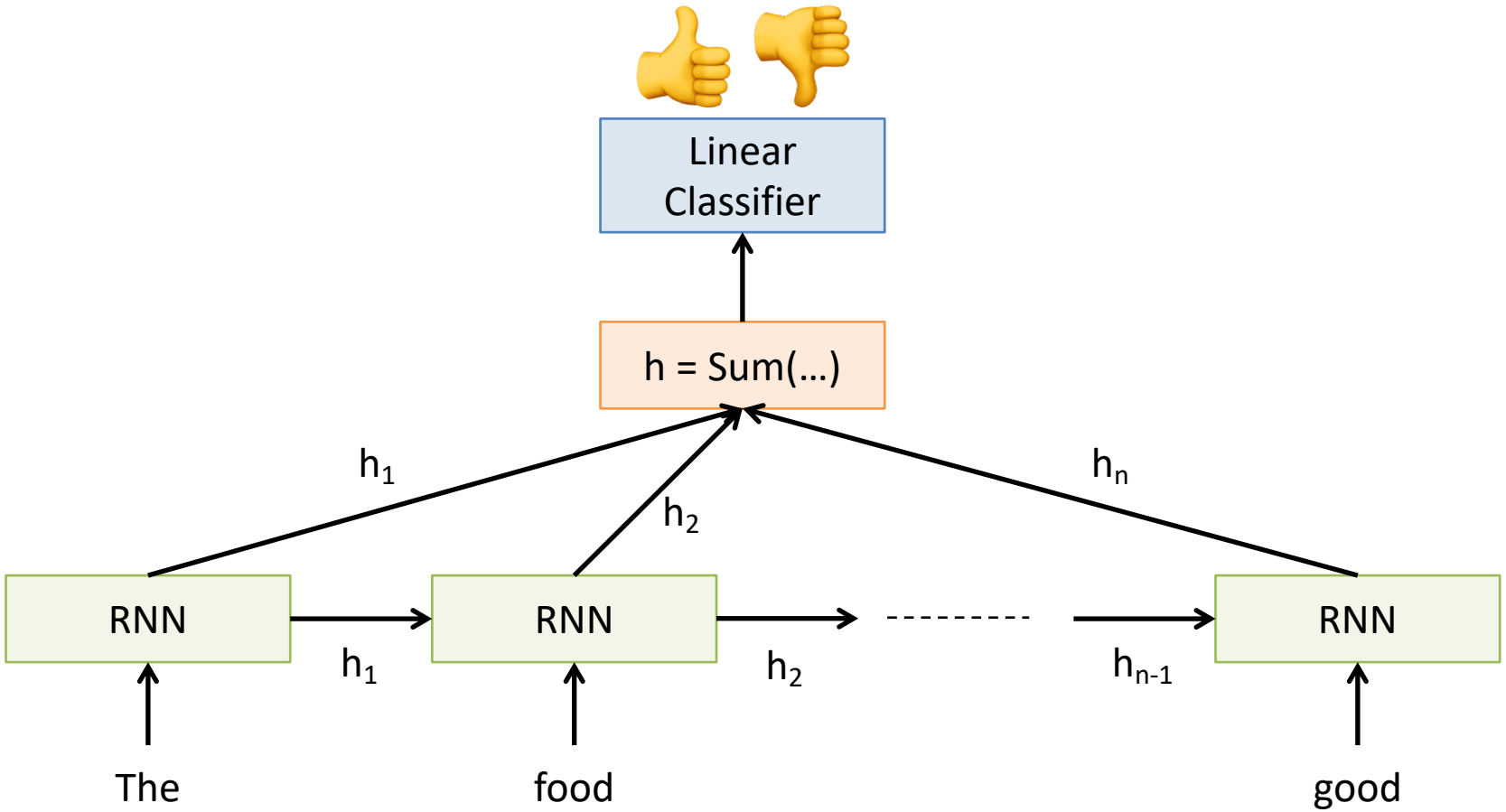
- Inputs: Multiple words, one or more sentences
- Outputs: Positive / Negative classification

Sentiment Classification



http://deeplearning.net/tutorial/lstm.html

Sentiment Classification

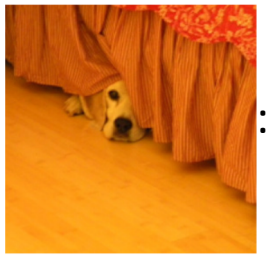


http://deeplearning.net/tutorial/lstm.html

Image Captioning

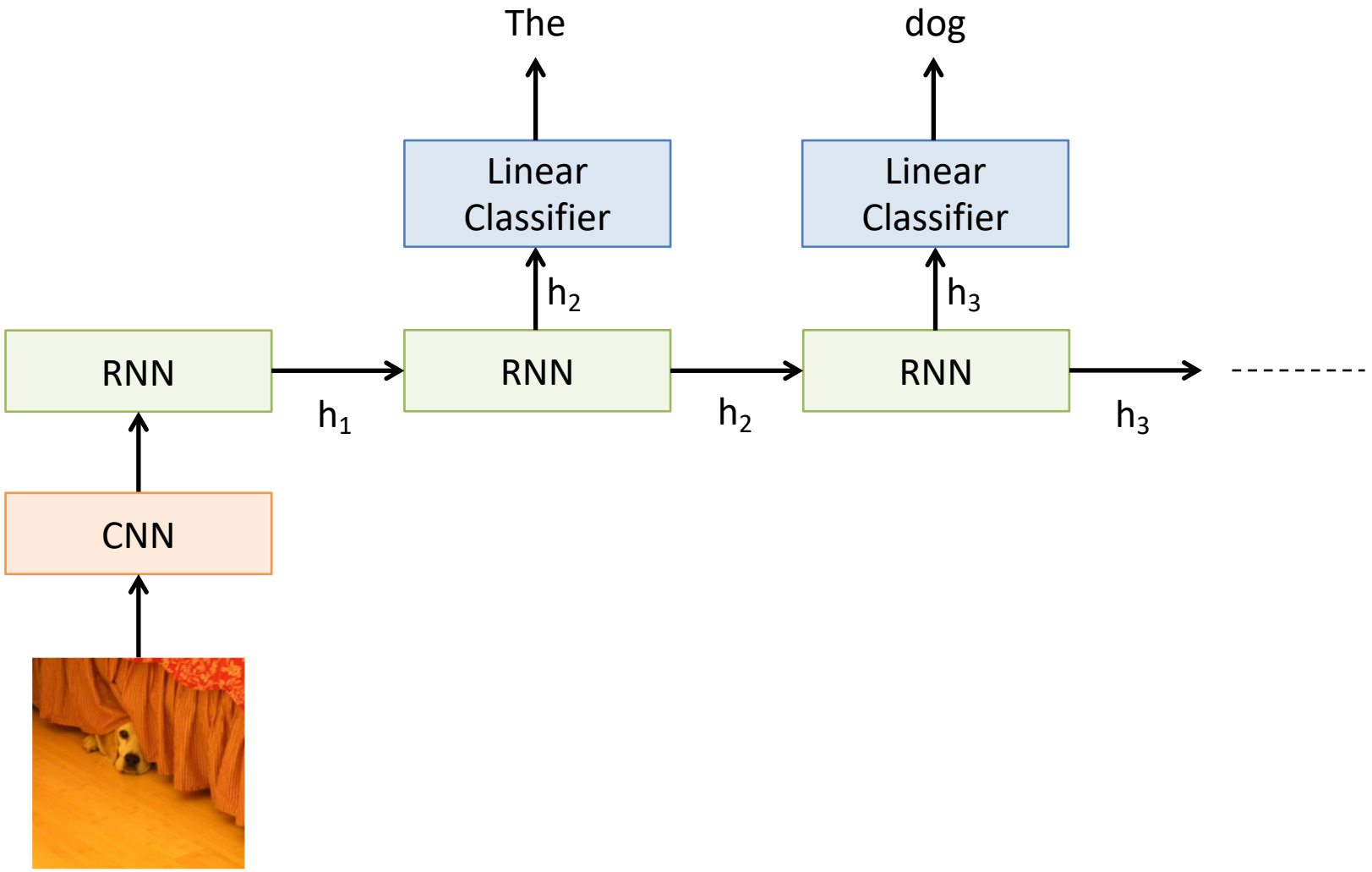
- Given an image, produce a sentence describing its contents

- Inputs: Image feature (from a CNN)
- Outputs: Multiple words (let's consider one sentence)



: The dog is hiding

Image Captioning



RNN Outputs: Image Captions

A person riding a motorcycle on a dirt road.

Two dogs play in the grass.

A herd of elephants walking across a dry grass field.

A group of young people playing a game of frisbee.

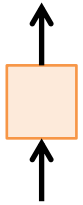
Two hockey players are fighting over the puck.

A close up of a cat laying on a couch.

[Show and Tell: A Neural Image Caption Generator, CVPR 15]

Input – Output Scenarios

Single - Single



Feed-forward Network

Single - Multiple

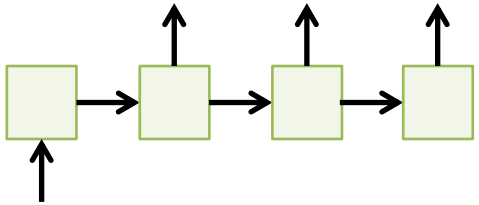
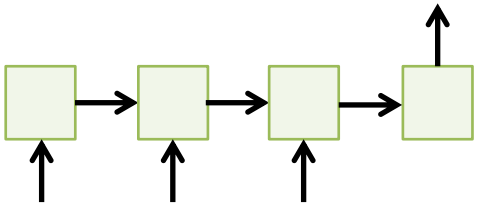


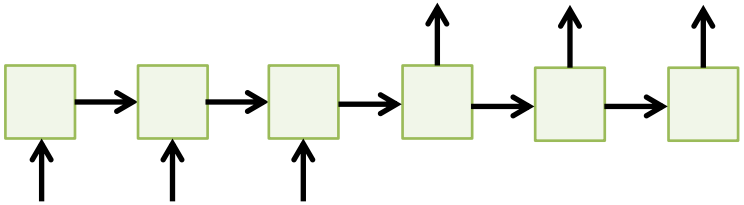
Image Captioning

Multiple - Single



Sentiment Classification

Multiple - Multiple



Translation

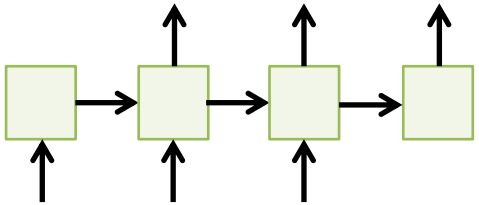


Image Captioning