There are a lot of different sequence models

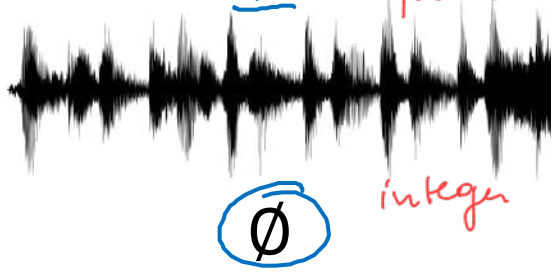
deeplearning.ai

Recurrent Neural Networks

Why sequence models?

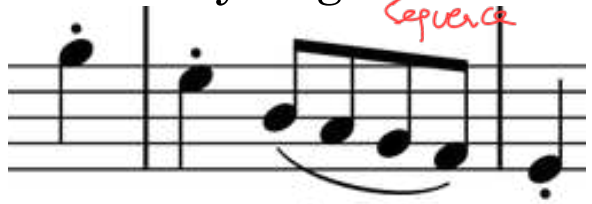
Examples of sequence data

Speech recognition



"The quick brown fox jumped over the lazy dog."

Music generation



Sentiment classification

"There is nothing to like in this movie."

DNA sequence analysis

AGCCCCTGTGAGGAACTAG

AGCCCCTGTGAGGAACTAG

Machine translation

Voulez-vous chanter avec moi?

Do you want to sing with me?

Video activity recognition

Running

Name entity recognition

Yesterday, Harry Potter met Hermione Granger.

Yesterday, Harry Potter met Hermione Granger.

Andrew Ng

deeplearning.ai

Recurrent Neural Networks

Notation

Motivating example

NLP

(x, y) = example and label

$T_x^{(i)}$ and $T_y^{(i)}$ = no. of tokens in i-th sample

$x^{(i)<t>}$ = t-th token in ith input sample

x: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ ..... $x^{<t>}$ ..... $x^{<9>}$

$T_x = 9$

y: 1 1 0 1 1 0 0 0 0

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ ..... $y^{<9>}$

$T_y = 9$

$X^{(i)<t>}$

$y^{(i)<t>}$

$T_x^{(i)} = 9$

$T_y^{(i)}$

15

Andrew Ng

Representing words

$x^{<t>}$

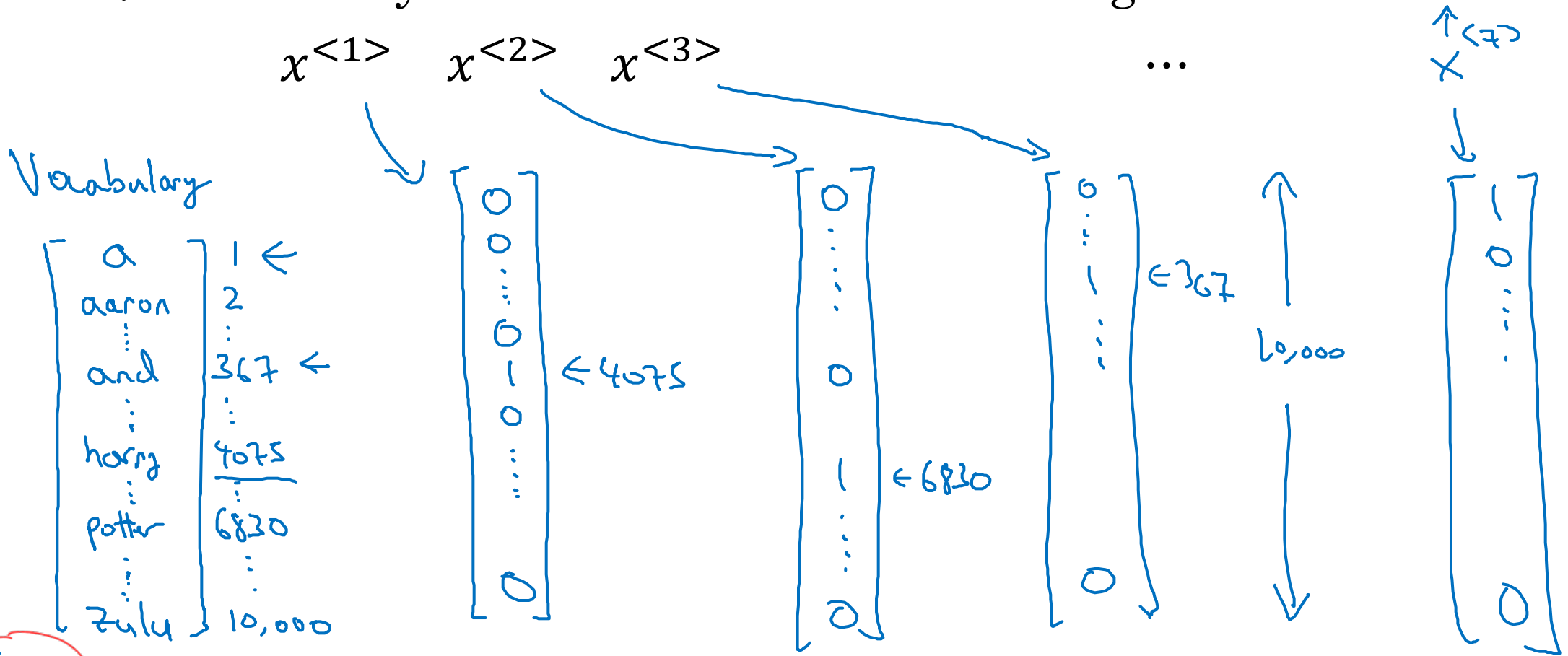
(x, y)

$x \rightarrow y$

In NLP, one way to encode a sample (sequence) is by one-hot encoding its tokens based on a vocabulary.

x: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ ... $x^{<7>}$ $x^{<9>}$



<UNK>

10,000 ← Vocab size in commercial applications is ~50k. Exceptional models arrive to 1M words.

Vocab entry to describe unknown words

One-hot

Representing words

x: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ … $x^{<9>}$

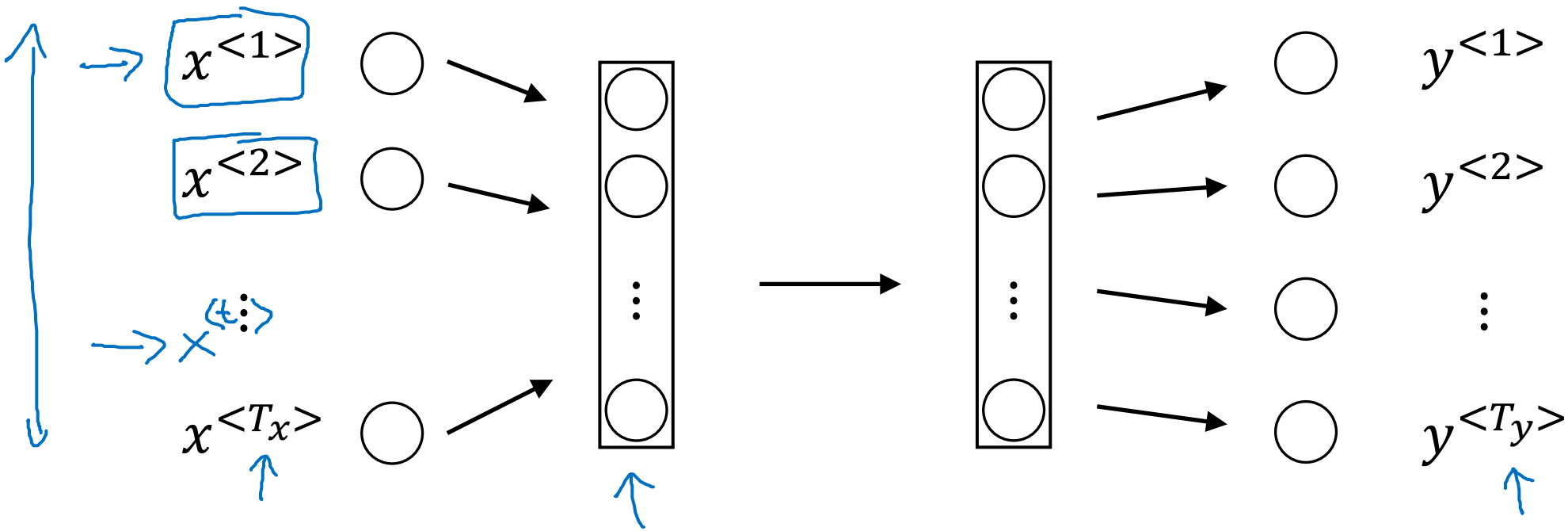
And = 367
Invented = 4700
A = 1
New = 5976
Spell = 8376
Harry = 4075
Potter = 6830
Hermione = 4200
Gran… = 4000

deeplearning.ai

Recurrent Neural Networks

Recurrent Neural Network Model

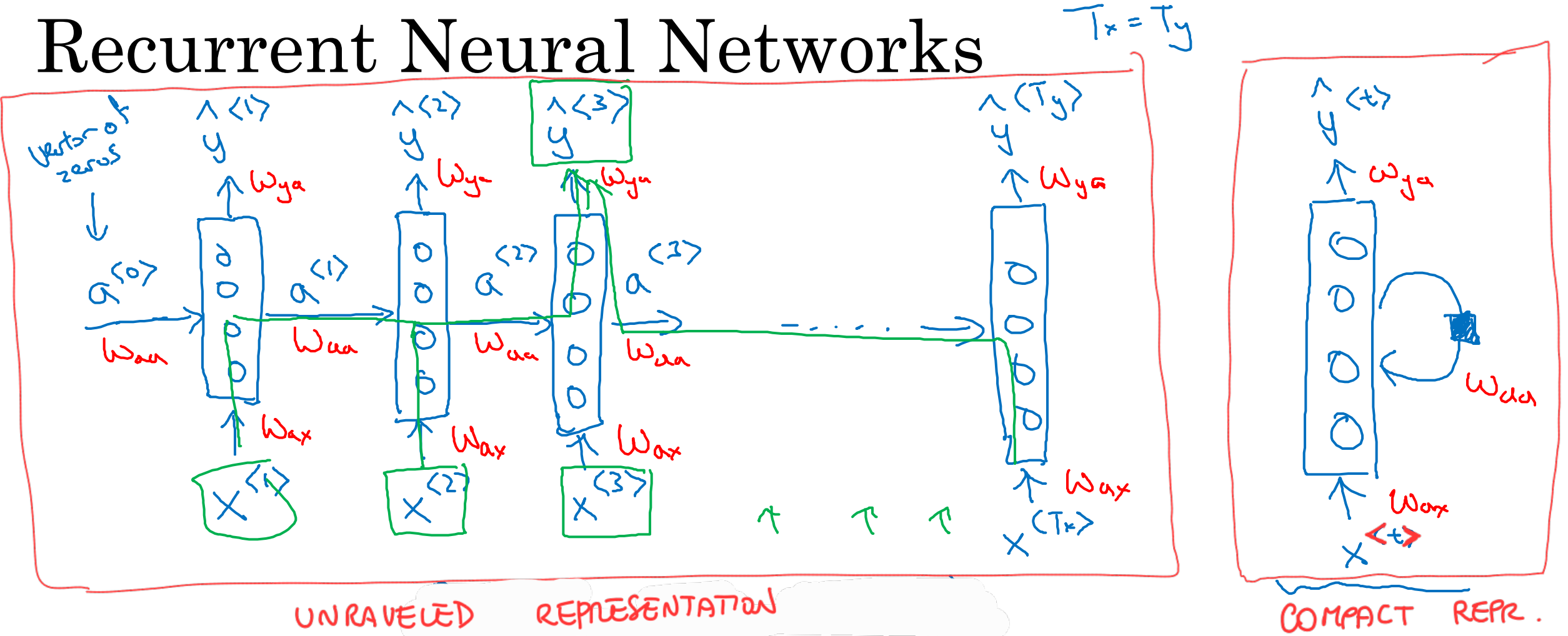
Why not a standard network?



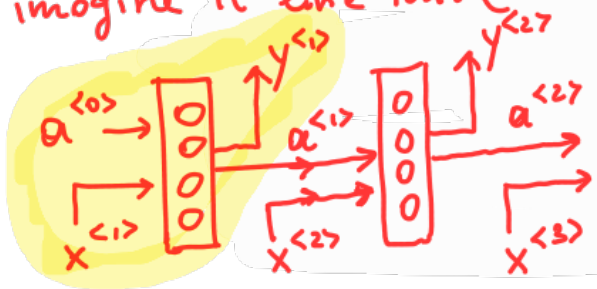
Problems:

- Inputs, outputs can be different lengths in different examples.
- Doesn't share features learned across different positions of text.

Recurrent Neural Networks



I imagine it like this (unrolled):



in yellow is the network

- the basic structure can be represented in unroveled (left) and compact (right) way.

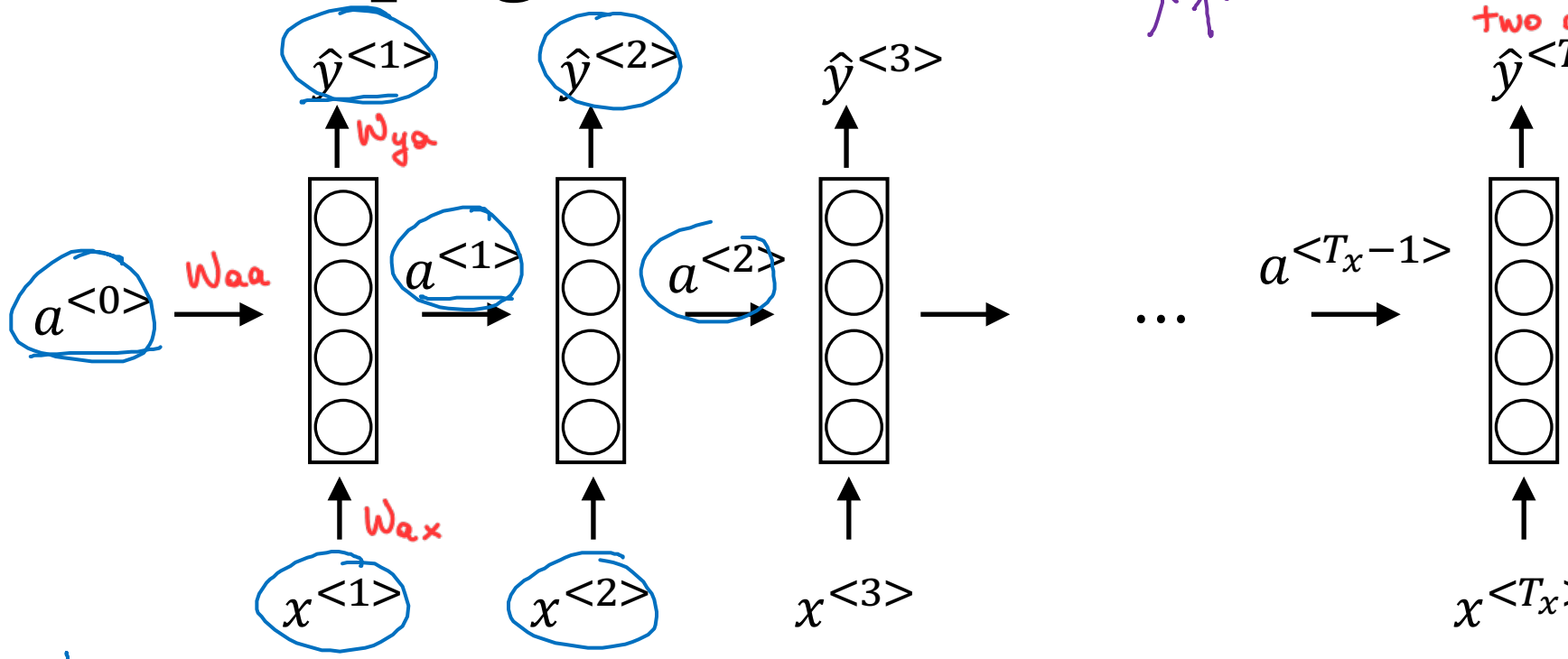
- Look at the compact representation. This is a RNN with 1 layer and 4 hidden units. (The non-RNN equivalent would be $x^{<i>} \rightarrow \begin{bmatrix} \circ \\ \circ \\ \circ \\ \circ \end{bmatrix} \rightarrow y^{<i>} = g\left(W_x x^{<i>} + b_x\right)$. The RNN peculiarity is that each hidden unit feeds back to itself: the activation of unit j at level k at time i, $a^{[k]\{j\}<i>}$, becomes one input of the same unit at time $i+1$.

NOTE THAT the unit has an "hidden state" $a^{<i>} = W_x x^{<i>} + b_x$ and its dimension (depends on the number of neurons) differs in general from the dimension of $x^{<i>}$. With one-hot encoding x can be in $\{0,1\}^{10000}$ but if the unit has only 4 neurons then the hidden state is also 4 dimensional (W is 4×10000).

- In the unroveled representation the same network is replicated horizontally at different points in time. The tail of the network is at the bottom, the head at the top.

Forward Propagation

$a \leftarrow W_{ax} x^{\langle 1 \rangle}$



The layer (it's only one in this case) computes two operations ($a^{\langle i \rangle}$ and $y^{\langle i \rangle}$).

Note that in non-RNNs the layer performed only one operation $y^{[i]} = g(a^{[i]})$.

Here two ops are needed because the activation is more complicated (not just Wx but $g(W_x x + W_a a + b_a)$)

$a^{\langle 0 \rangle} = \vec{0}$

$$a^{\langle 1 \rangle} = g_1(W_{aa} a^{\langle 0 \rangle} + W_{ax} x^{\langle 1 \rangle} + b_a) \leftarrow \tanh / \text{ReLU}$$

$$\hat{y}^{\langle 1 \rangle} = g_2(W_{ya} a^{\langle 1 \rangle} + b_y) \leftarrow \text{Sigmoid}$$

$$a^{\langle t \rangle} = g(W_{aa} a^{\langle t-1 \rangle} + W_{ax} x^{\langle t \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = g(W_{ya} a^{\langle t \rangle} + b_y)$$

Andrew Ng

Simplified RNN notation

$$a^{<t>} = g(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

(100, 100) 100 (100, 10,000) 10,000

$$\hat{y}^{<t>} = g(W_{ya}a^{<t>} + b_y)$$

$$\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$$

by stacking the layer's inputs at time t, one gets the simplified notation:

$$a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$$

$$\begin{bmatrix} W_{aa} ; W_{ax} \end{bmatrix} = W_a$$

100 × (100, 10000)

$$[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix}$$

100 + 10000 = 10100

$$[W_{aa} ; W_{ax}] \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa}a^{<t-1>} + W_{ax}x^{<t>}$$

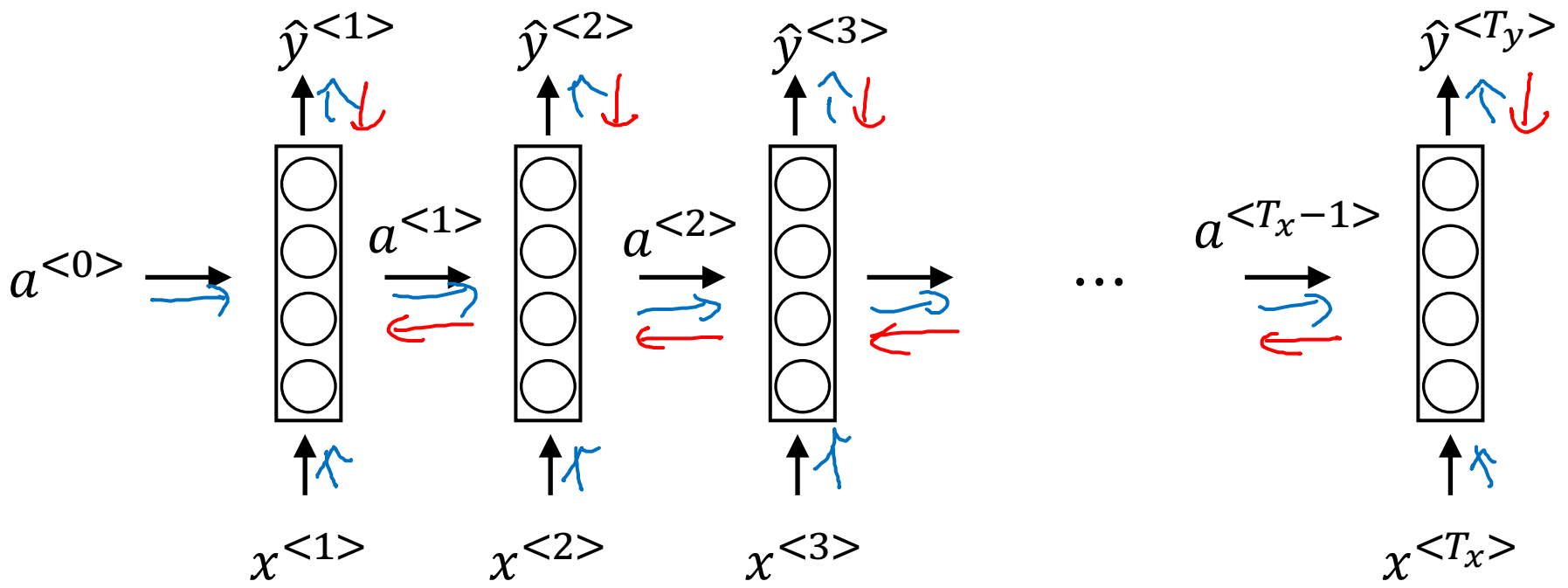
Andrew Ng

deeplearning.ai

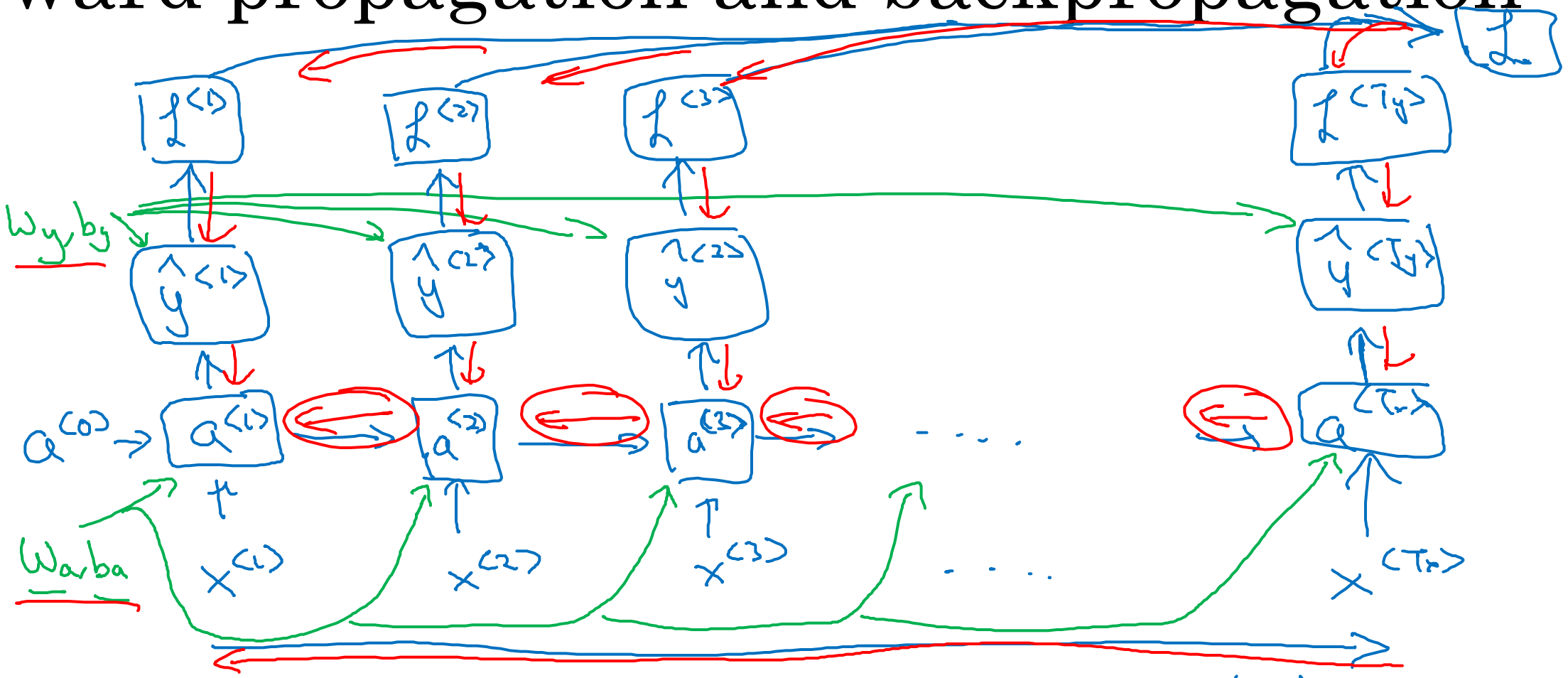
Recurrent Neural Networks

Backpropagation through time

Forward propagation and backpropagation



Forward propagation and backpropagation



$$\mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>}\log\hat{y}^{<t>} - (1-y^{<t>})\log(1-\hat{y}^{<t>})$$

$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

OVERALL LOSS

LOSS FOR ELEMENT T OF THE INPUT SEQUENCE

The loss for a certain input sequence is the summed loss of the single inputs of the sequence.

Backpropagation through time

Andrew Ng

deeplearning.ai

Recurrent Neural Networks

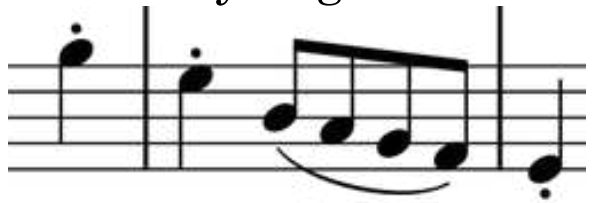
Different types of RNNs

Different architecture to tackle different types of problems. The main difference between problems is the number of inputs and outputs.

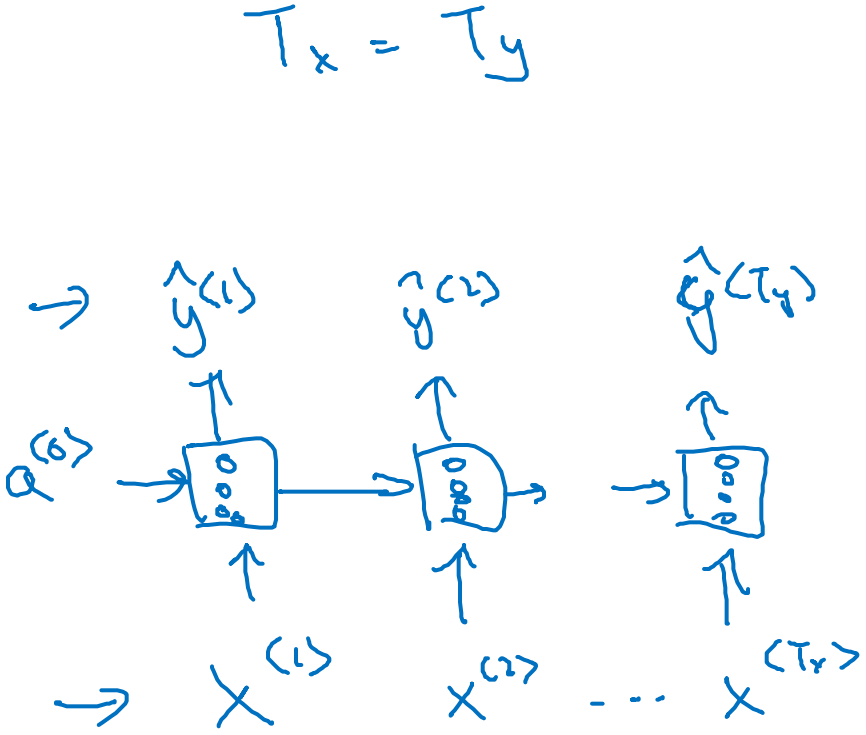
Examples of sequence data

Tx Ty

x y

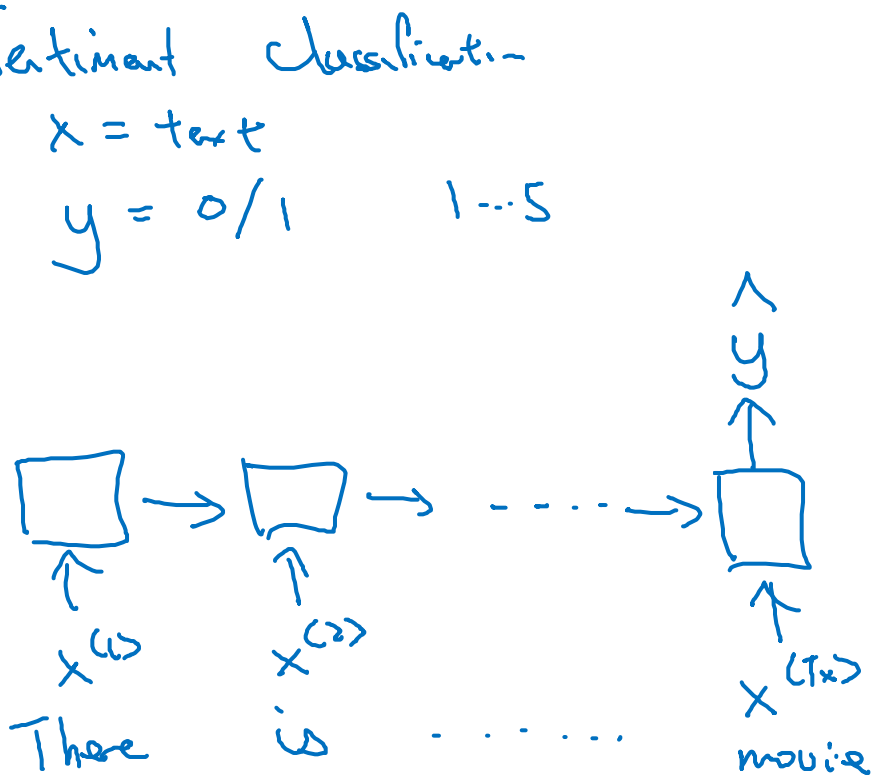
Speech recognition	 ∅	→	"The quick brown fox jumped over the lazy dog."
Music generation		→	
Sentiment classification	"There is nothing to like in this movie."	→	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger.

Examples of RNN architectures



Many-to-many

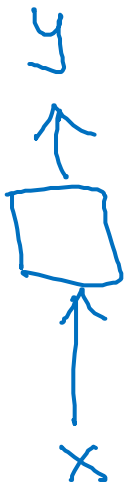
- many inputs and many outputs
- no inputs = no outputs



Many-to-one

- many inputs
- only one output

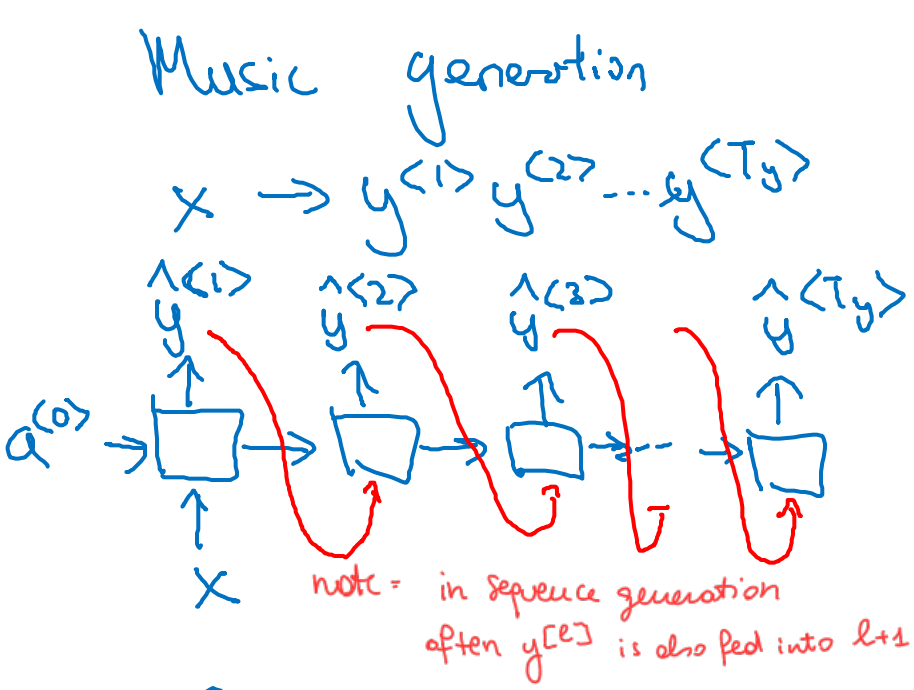
In this frame, a standard NN can be defined as one-to-one



One-to-one

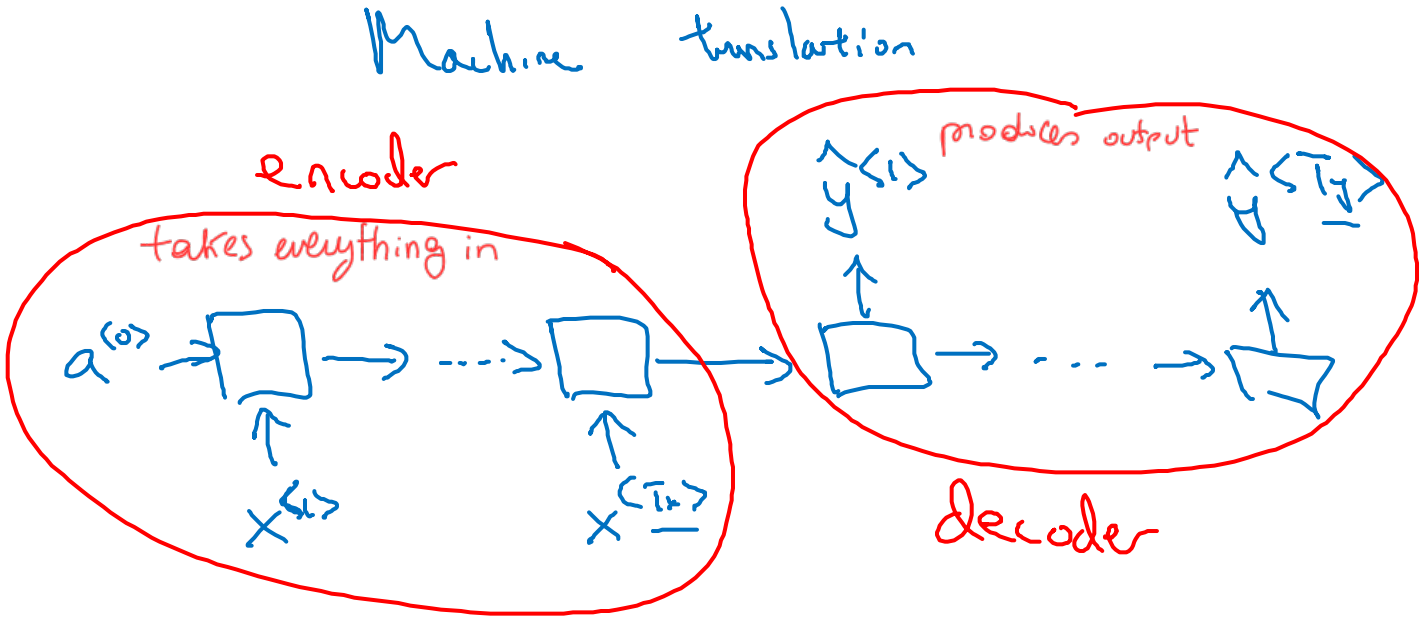
Andrew Ng

Examples of RNN architectures



One-to-many

$x = \phi$



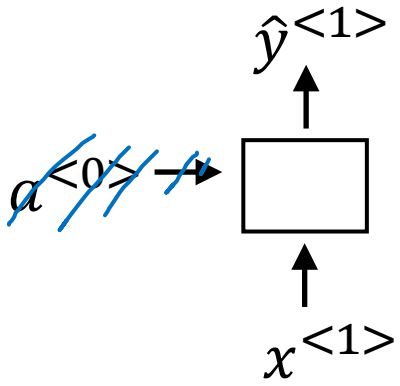
Many-to-many

different number of inputs and outputs

A FINAL TYPE OF ARCHITECTURE IS THE TRANSFORMER

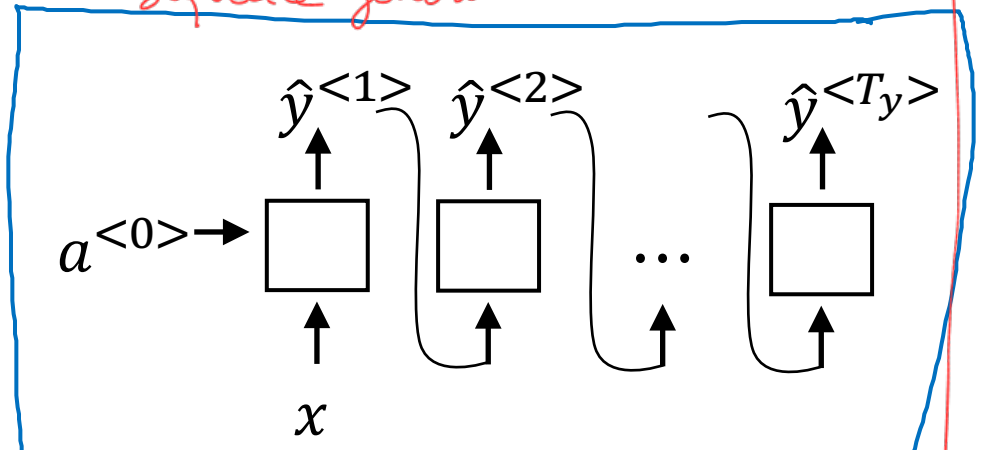
Summary of RNN types

Standard NN



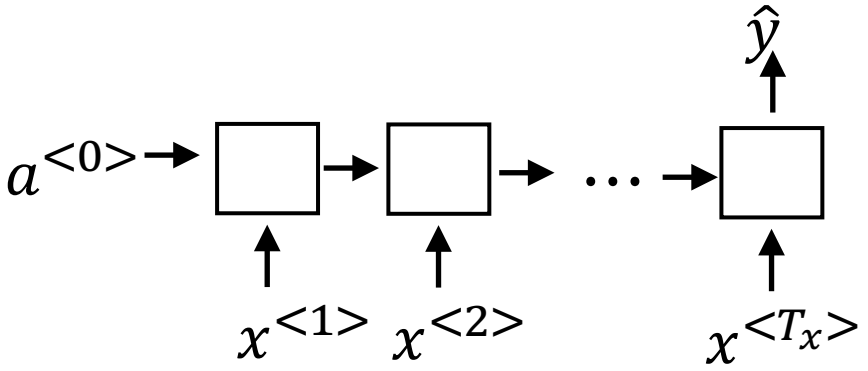
One to one

Sequence generation

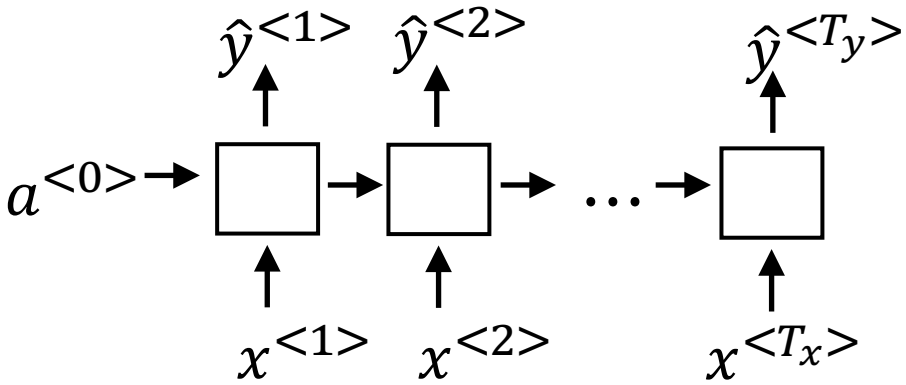


One to many

eg sentiment analysis



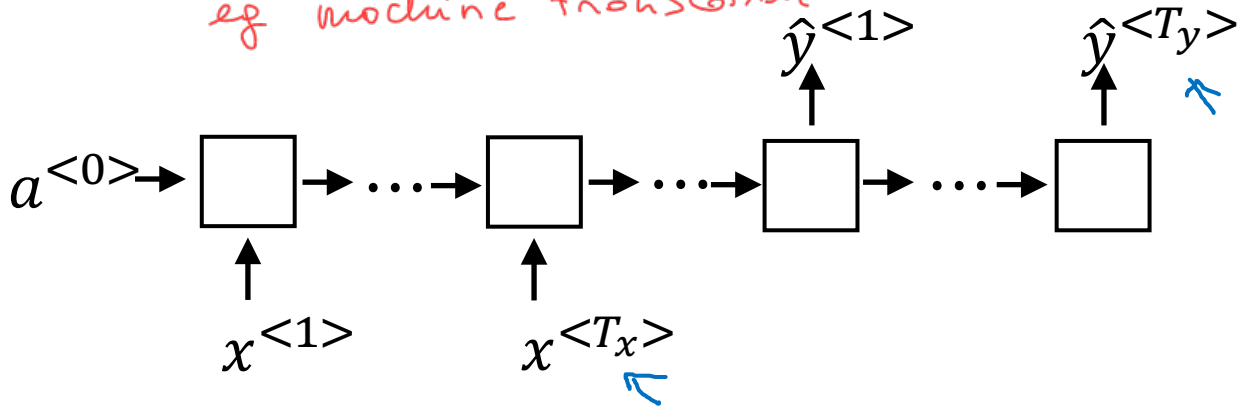
Many to one



Many to many

$T_x = T_y$

eg machine translation



Many to many

Andrew Ng

deeplearning.ai

Recurrent Neural Networks

Language model and sequence generation

What is language modelling?

Language modelling = establishing the probability of a sequence of words for a certain language.

Speech recognition

The apple and pair salad.

→ The apple and pear salad.

P(The apple and pair salad) = 3.2×10^{-13}

P(The apple and pear salad) = 5.7×10^{-10}

$P(\text{Sentence}) = ?$ $\quad P(y^{\langle 1 \rangle}, y^{\langle 2 \rangle}, \ldots, y^{\langle T_y \rangle})$

Language modelling with an RNN

The first step is to tokenize the sentence = mapping each element of the sentence with an individual token (element of the vocabulary) of the language.

Training set: large corpus of english text.

Tokenize

The vocabulary can include special tokens like <UNK>, <EOL>, <.>, ...

Cats average 15 hours of sleep a day. <EOS>

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ ... $y^{<8>}$ $y^{<9>}$

$x^{<t>} = y^{<t-1>}$

The Egyptian ~~Mau~~ is a bread of cat. <EOS>

<UNK>

10,000

Language modelling task

- DEFINITION: determining the probability of a series of words in a certain language.

- STEP 1: TOKENIZATION AND MAPPING: Isolating the relevant elements in the input sentence (words, relevant punctuation, EOL if desired, etc), that is the "tokens", and mapping each of them with an entry of the vocabulary of the given language.

- STEP 2: PREDICTING PROBABILITY of the sequence of words. (Andrew Ng provided a somewhat confusing explanation).

For each token in the sequence, the network outputs in order

$$P(1^{st}\text{ token}),\ P(2^{nd}\text{ tok} \mid 1^{st}\text{ tok}),\ \dots,\ P(t^{th}\text{ tok} \mid (t-1)^{th}\text{ tok})$$

that is

$$\hat{y}^{<1>} = P(y^{<1>}),\ \hat{y}^{<2>} = P(y^{<2>} \mid y^{<1>}),\ \dots,\ P(y^{<t>} \mid y^{<1>}, y^{<2>}, \dots, y^{<t-1>}).$$

The probability of the sentence is obtained as the product of those probabilities (... although this assumes independence of the elements in the sequence ...).

$$P(y^{<1>}, y^{<2>}, \dots, y^{<t>}) = P(y^{<1>}) \cdot P(y^{<2>} \mid y^{<1>}) \cdot \dots \cdot P(y^{<t>} \mid y^{<1>}, y^{<2>}, \dots, y^{<t-1>})$$

$$= \hat{y}^{<1>} \cdot \hat{y}^{<2>} \cdot \dots \cdot \hat{y}^{<t>}$$

- ACTUALLY: the NN does not return only one probability but rather a softmax over all the elements of the vocabulary

$$\hat{y}^{<1>} = [P(v_1), P(v_2), \dots, P(v_{10000})],\ \hat{y}^{<2>} = \left[P(v_1 \mid y^{<1>}), \dots, P(v_{10000} \mid y^{<1>})\right],\ \hat{y}^{<3>} = \dots.$$

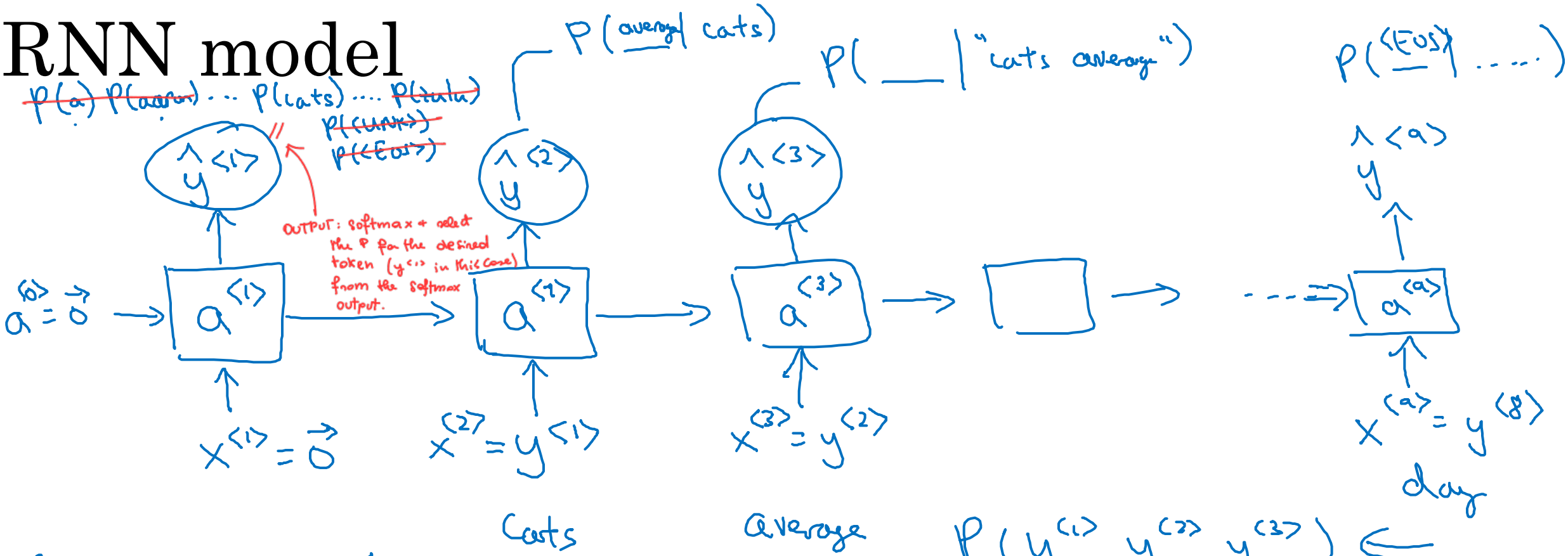
(although Andrew Ng writes $\hat{y}^{<t>} = v_i \overset{word}{s.t.} \max\left([P(v_i \mid y^{<1>}, \dots, y^{<t-1>})]_{i=1}^{10000}\right)$.)

- the NN's cost function uses a CROSS entropy loss that compares the predicted softmax to the ground truth probability distrib of the desired word wrt vocabulary.

example The cat is on the table

$$y^{<2>} = \text{cat} \dots \text{actually} \dots = [0 \dots \underset{\text{prob of cat}}{1} 0 \dots 0 \underset{\text{prob of Aaron}}{}],\ \hat{y}^{<2>} = [10^{-13} \dots 10^{-4}\ 0.6\ 10^{-2} \dots 10^{-6}],\ L^{<2>} = \sum_{i=1}^{10000} y_i^{<2>} \cdot \log \hat{y}_i^{<2>}$$

RNN model



Cats average 15 hours of sleep a day. <EOS>

$$\mathcal{L}(\hat{y}^{<t>}, y^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$$

CROSS ENTROPY LOSS $(\hat{y}^{<t>}, y^{<t>})$

$$\mathcal{L} = \sum_t \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

COST FUNCTION

$$P(y^{<1>}, y^{<2>}, y^{<3>}) = P(y^{<1>})\, P(y^{<2>} \mid y^{<1>})\, P(y^{<3>} \mid y^{<1>}, y^{<2>})$$

Andrew Ng

Once the NN has been fitted to a language model, it can be used for sampling a sequence:

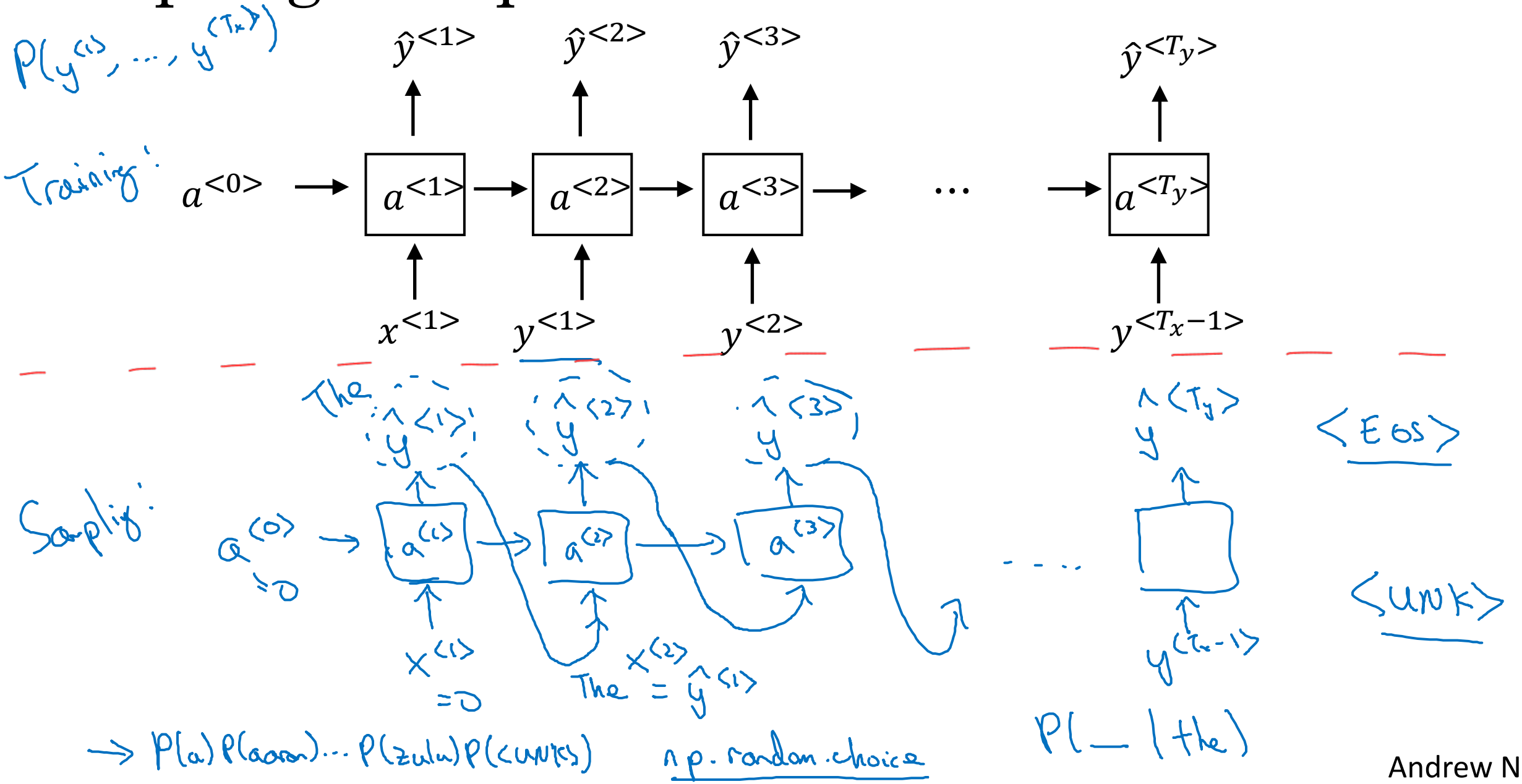
- interpret the output: $output^{<t>}$ = vocab. entry corresponding to $\max(\hat{y}^{<t>})$. eg. "cat"
- use the interpreted input as new entry in the NN: $x^{<t+1>} = output^{<t>}$
- terminate the generation after a fixed number of tokens or when <EOL> is generated.
- implement fallback strategies for when <UNK> is generated.

deeplearning.ai

Recurrent Neural Networks

Sampling novel sequences

Sampling a sequence from a trained RNN



Andrew Ng

Character-level language model

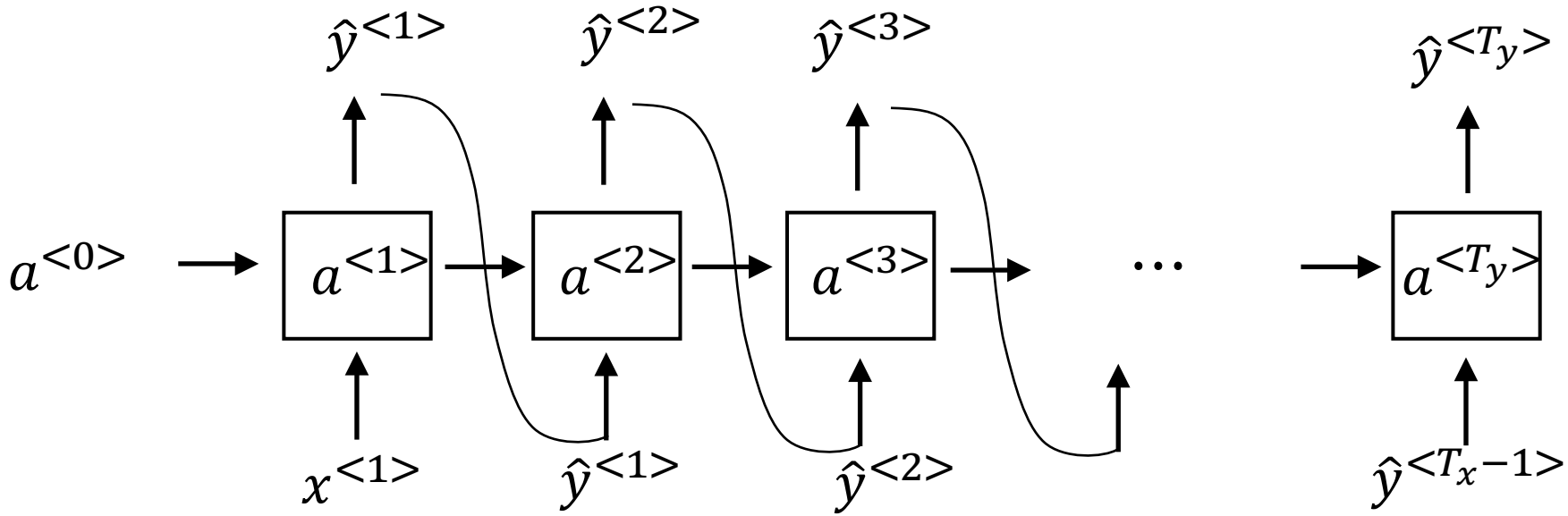
(as opposed to language-level language model). One advantage is not requiring an <UNK> character, but the big disadvantage is having to be trained on much longer sequences.

→ Vocabulary = [a, aaron, …, zulu, <UNK>]

→ Vocabulary = [a, b, c, … , z, ␣, ., ,, ;, 0, …, 9, A, …, Z]

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$

Cat average … Mau



Andrew Ng

Sequence generation

News

President enrique peña nieto, announced sench's sulk former coming football langston paring.

"I was not at all surprised," said hich langston.

"Concussion epidemic", to be examined.

The gray football the told some and this has on the uefa icon, should money as.

Shakespeare

The mortal moon hath her eclipse in love.

And subject of this thou art another this fold.

When besser be my love to me see sabl's.

For whose are ruse of mine eyes heaves.

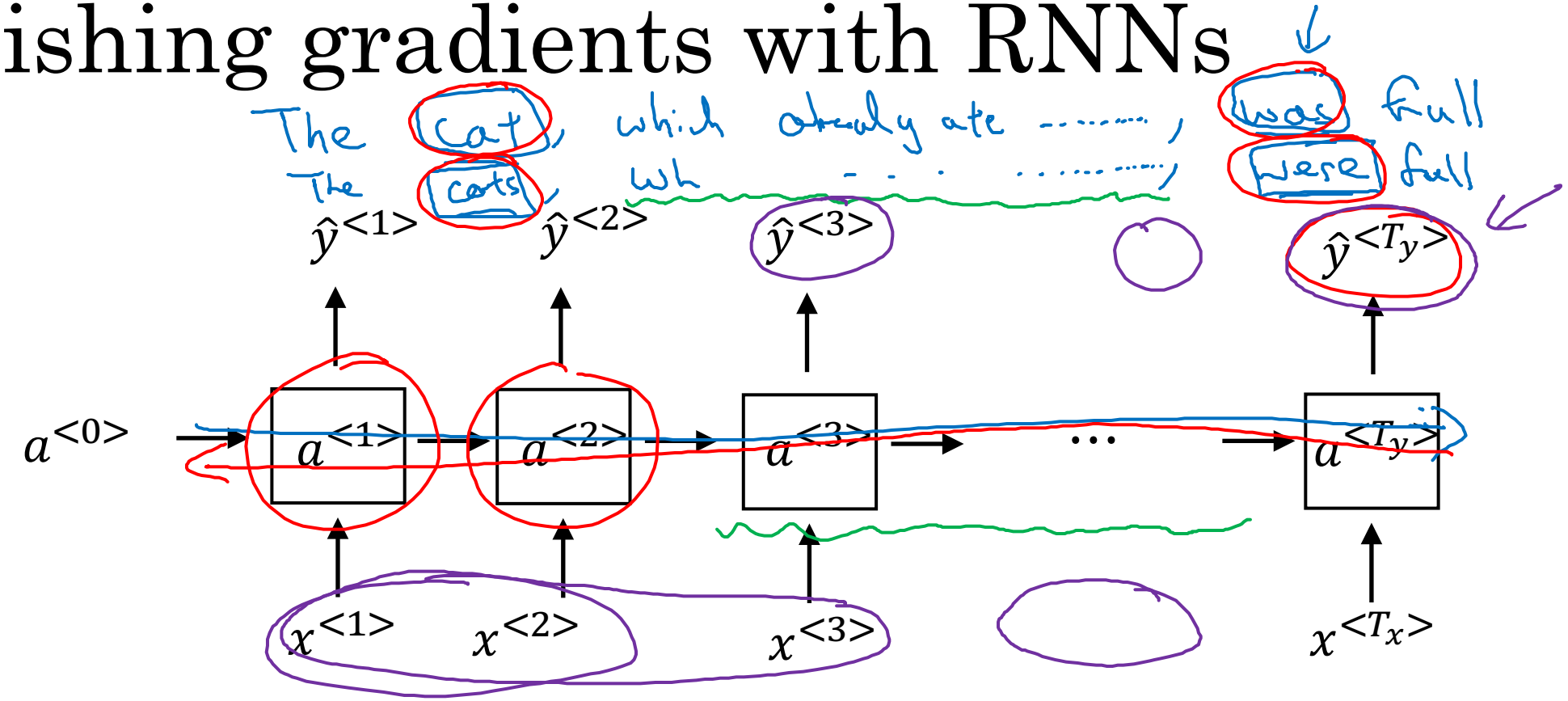
deeplearning.ai

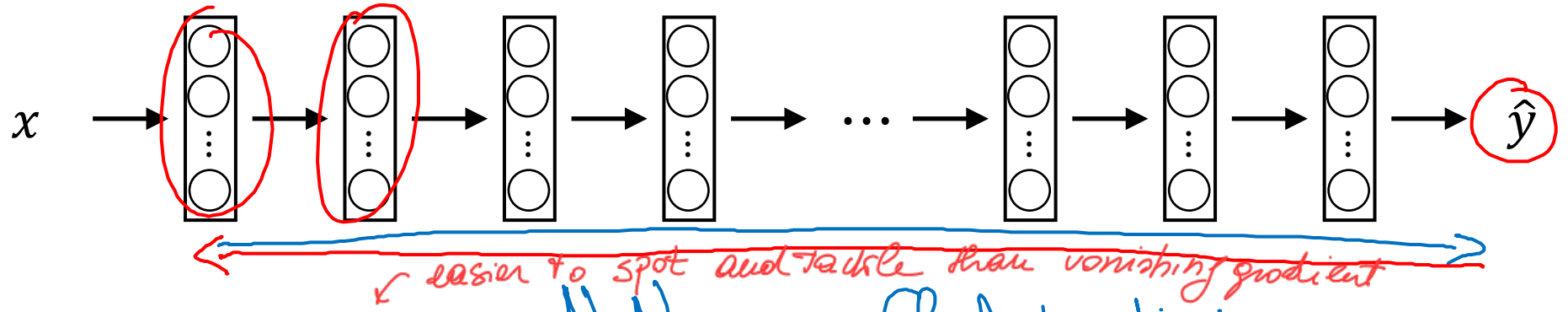
Recurrent Neural Networks

Vanishing gradients with RNNs

RNNs are prone to vanishing gradients in case they are used to handle long sequences.
=> RNNs are not at capturing long term dependencies

Vanishing gradients with RNNs





Exploding gradients.

NaN

Gradient clipping

Recurrent Neural Networks

Gated Recurrent Unit (GRU)

A GRU is a modification of the standard Recurrent Unit that produces outputs (activations and updates) with some "adaptive momentum". The activation at time $<t>$ is the weighted sum of the hidden state $\tilde{c}^{<t>}$ of the unit and the old activation $c^{<t-1>}$ using a "gate function" as weight. The gait function Γ depends from $c^{<t-1>}$ and $x^{<t>}$.

$$c^{<t>} = \Gamma * c^{<t>} + (1-\Gamma) * c^{<t-1>}$$

The idea is that Γ allows to keep track of relevant tokens across very long sentences. Γ is set (automatically) close to 1 when the input is important for the sentence and then is near zero for the remaining terms until another important word appears. When Γ is zero, it forces the output to strongly relate to the word for which it was last set to 1.

"The cat, which ate the soup and the apple, was fat and the child ..."

$\Gamma = 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1$

deeplearning.ai

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1-\Gamma) * c^{<t-1>}$$

$$\tilde{c}^{<t>} = \tanh\left(W_c \cdot [c^{<t-1>}, x^{<t>}] + b_c\right)$$

$$\Gamma_u = \sigma\left(W_u \cdot [c^{<t-1>}, x^{<t>}] + b_u\right)$$

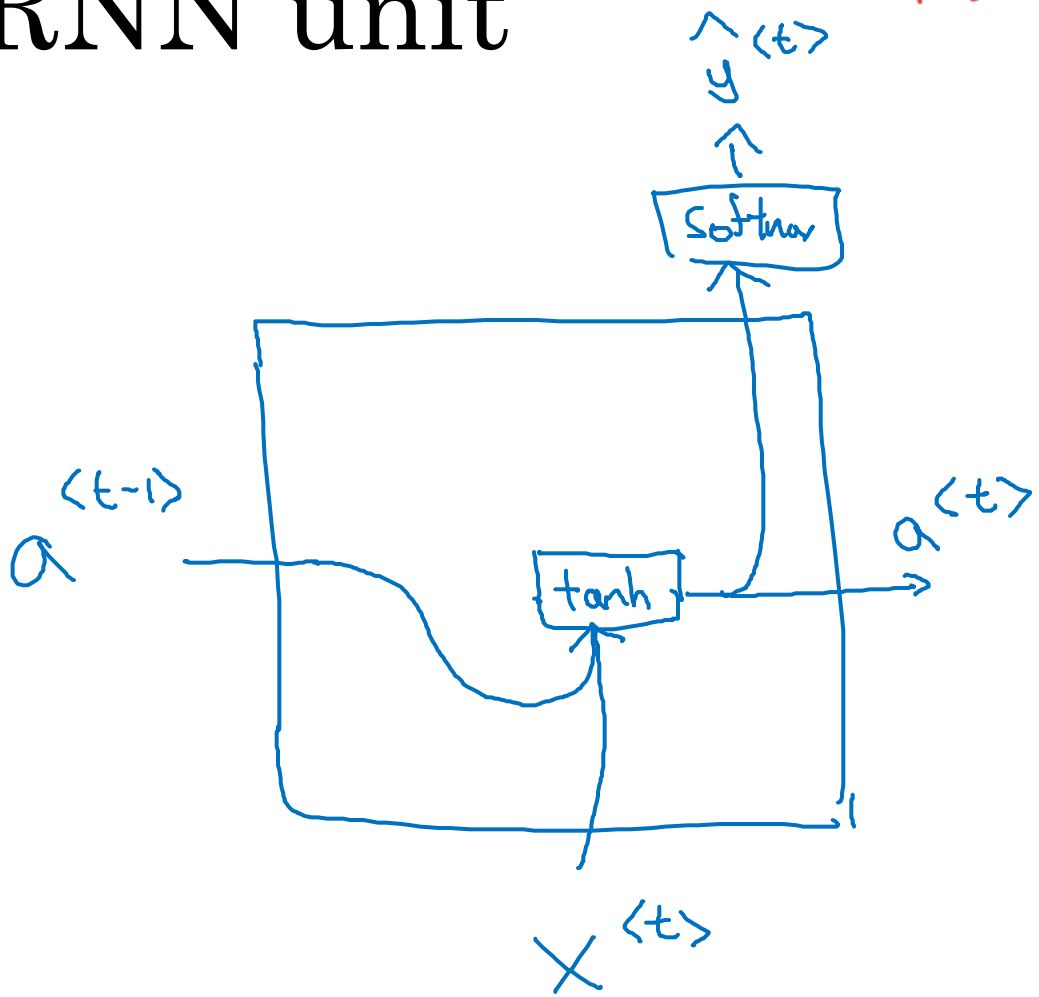
NOTE: $c^{<t>} = a^{<t>}$ for GRU but this is not the case for LSTM units.

NOTE: $\tilde{c}^{<t>}$

update gate: in more advanced GRUs there are more types of gates. An example is the relevance gate Γ_r, which helps forgetting the previous hidden state (unclear).

NOTE: GRU come after LSTM, as a semplification of LSTM and often work as well.

RNN unit



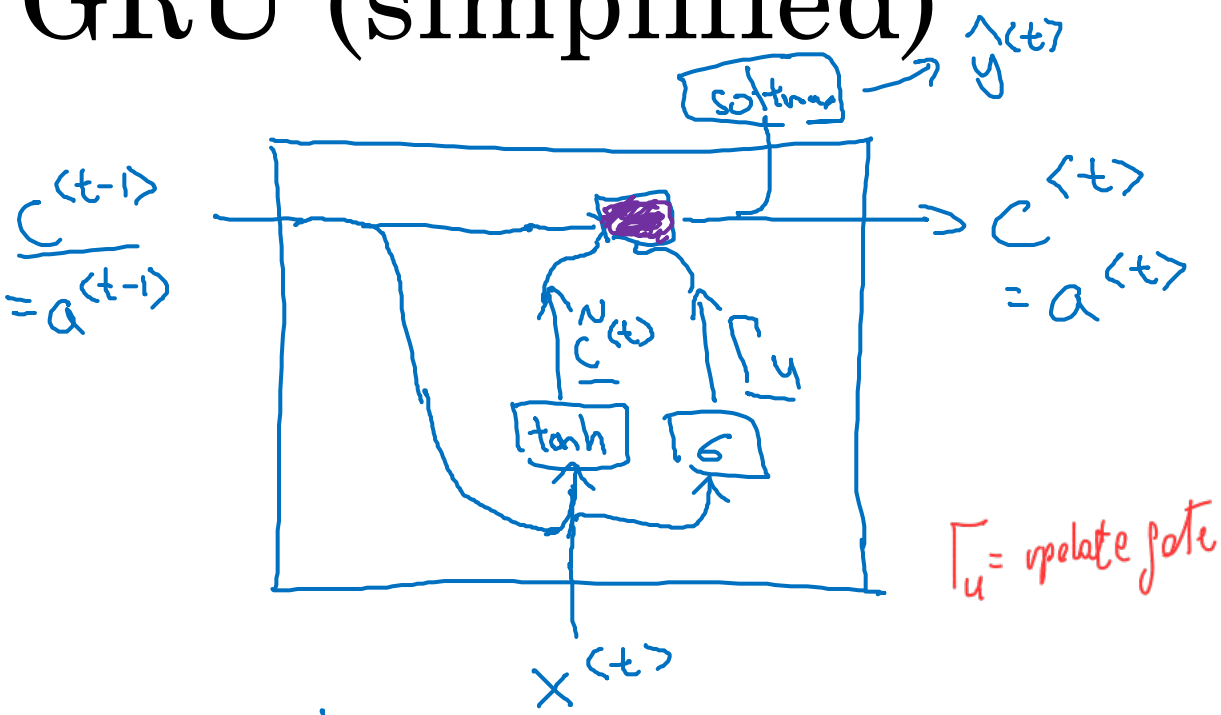
TLDR: differently from RU, the activation $c^{<t>}$ is mixed with the old activations using a gate function as weight. The gate function allows to maintain the influence of old activations high for a long time.

In practice it separates the hidden state ($\tilde{c}$) from the actual activation c.

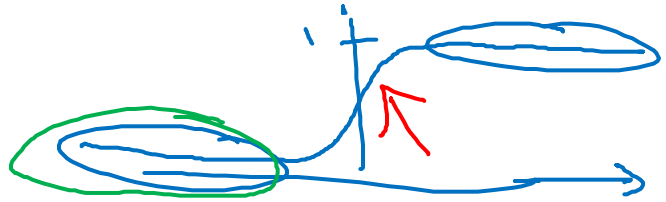
tanh

$$a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}] + b_a)$$

GRU (simplified)



Γ_u = update gate



c = memory cell

$c^{\langle t \rangle} = a^{\langle t \rangle}$

$\tilde{c}^{\langle t \rangle} = \tanh\left(W_c \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_c\right)$

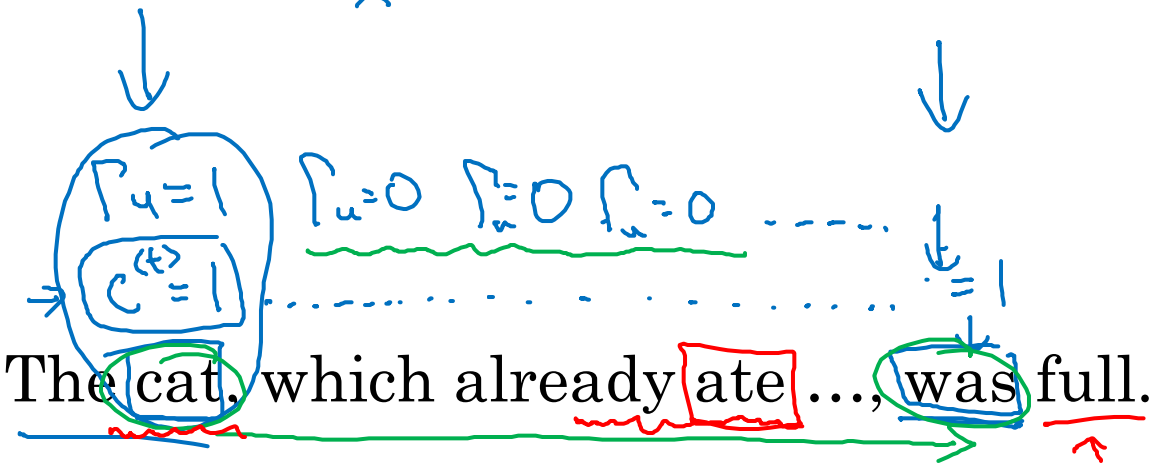
$\Gamma_u = \sigma\left(W_u \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_u\right)$

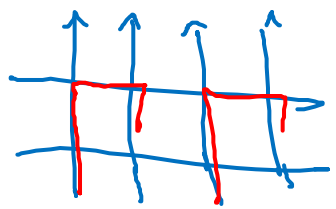
"update"

$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) * c^{\langle t-1 \rangle}$

element-wise

Gate





$\Gamma_u = 0.000001$

→ The cat, which already ate …, was full.

[Cho et al., 2014. On the properties of neural machine translation: Encoder-decoder approaches]

[Chung et al., 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling]

Full GRU

Γ_r = relevance gate

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$\tilde{c}^{<t>} \leftarrow \tanh\left(\left[\Gamma_r * c^{<t-1>}, x^{<t>}\right] + b_c\right)$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$$

~~LSTM~~

IGNORED IN LSTMS !

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) + c^{<t-1>}$$

alternative notation in literature

The cat, which ate already, was full.

LSTM units are a more general version of GRU that have 3 gates (input, forget, output) instead of just one (update).
↑ NG calls it update
The idea is to create a separation between the input of the cell and the output in the form of an extra memory cell $c^{<t>}$ called "cell state", which is distinct from the output activation (in GRU the output = $a^{<t>} = c^{<t>} = \Gamma \tilde{c}^{<t>} + (1-\Gamma) c^{<t-1>}$). So in GRU we have hidden state $\tilde{c}$ and activation $c = a$, while in LSTM we have hidden state, cell state and output activation.

- LSTM is more general because has 2 gates (Γ_i, Γ_f) that regulate the information flow in and out of the cell state c (while GRU only has one Γ_u to regulate the input of new info $\tilde{c}$ into the activation $c = a$).
- Moreover LSTM has a dedicated forgetting gate Γ_f to regulate the contribution of memory (old cell state $c^{<t-1>}$) to the current cell state. (Actually they call it forget gate, but "memory gate" would be more precise).

Recurrent Neural Networks

deeplearning.ai

TLDR: • more general/flexible version of GRU (also older). SEE FORMULAS IN 2 SLIDES

- it has 3 gates (input = update, output, forget) instead of only one (update).
- separate "cell state" memory cell from hidden state and output.
- 2 gates (in, out) regulate information flow through cell state.
- 1 gate (update) regulate influence of old cell state on new one.

LSTM (long short term memory) unit

- in general one should try both GRU (lighter) and LSTM. If no time to check, pick LSTM.

GRU and LSTM

GRU vs LSTM

GRU

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

$$a^{<t>} = c^{<t>}$$

Γ_f

LSTM

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

"INPUT" (update)
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

(forget)
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

(output)
$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$$

[Hochreiter & Schmidhuber 1997. Long short-term memory]

Andrew Ng

LSTM units

GRU

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[\, c^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_r = \sigma(W_r[\, c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

$$a^{<t>} = c^{<t>}$$

LSTM

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * c^{<t>}$$

[Hochreiter & Schmidhuber 1997. Long short-term memory]

LSTM in pictures

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

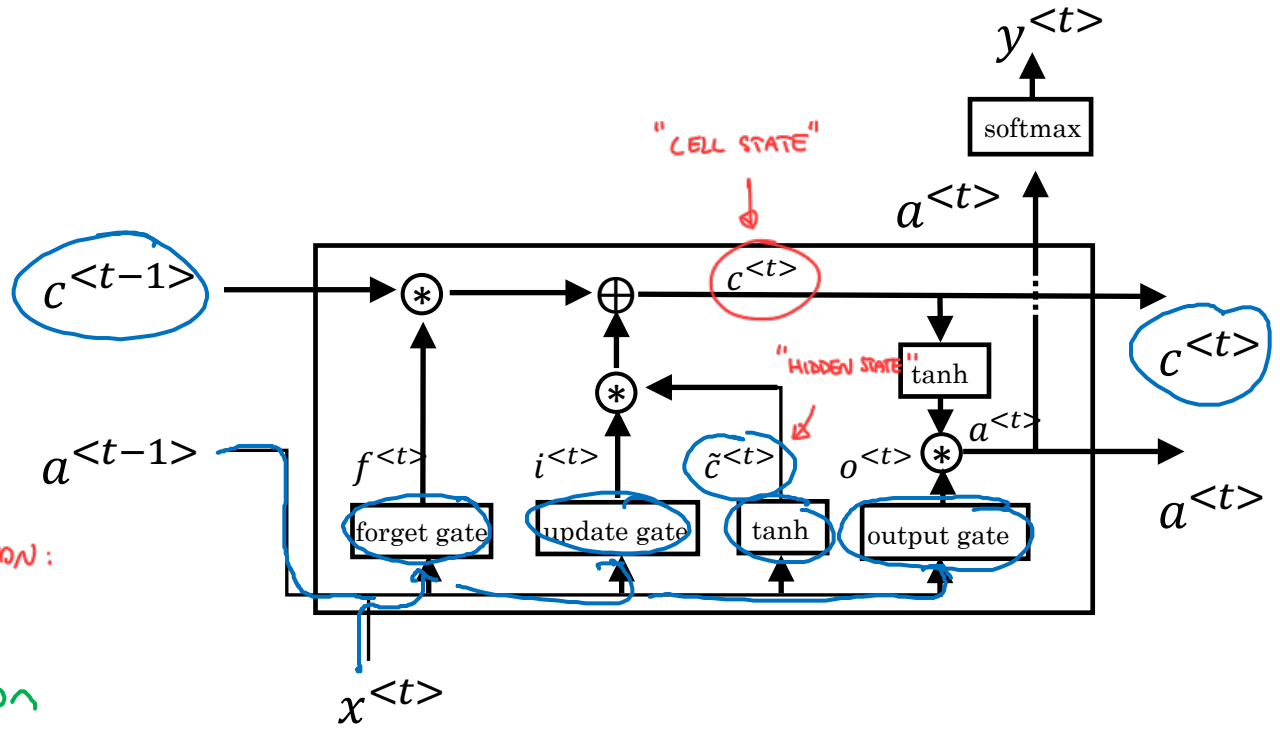
$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

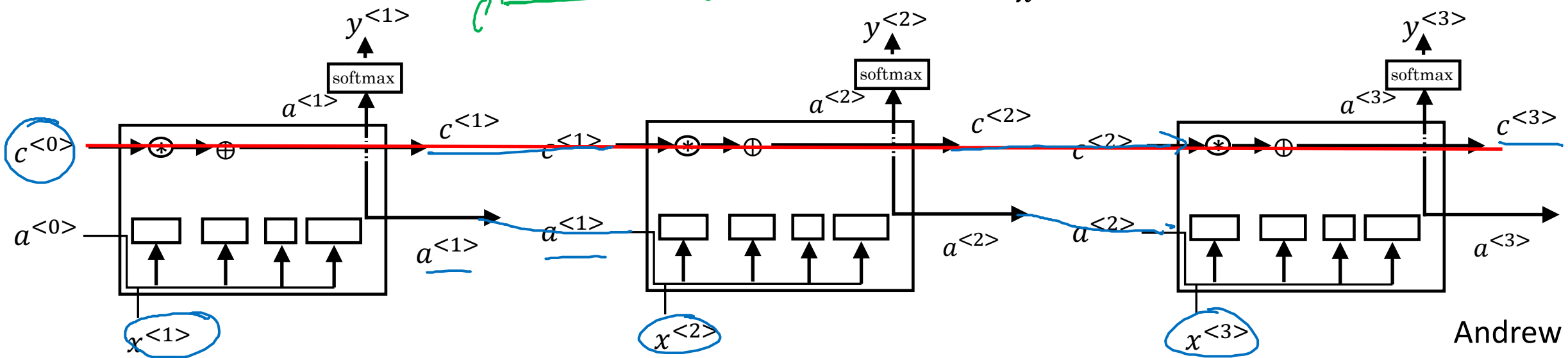
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * c^{<t>}$$

$c^{<t-1>}$

COMMON VARIATION: peephole connection





Andrew Ng

BRNN are used to make predictions also based to information in the future (for example useful for named entity recognition). They do so:

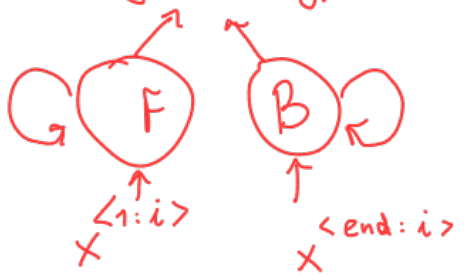
- by having access to the entire input sequence (no online)
- by having one "forward" recurrent unit that processes the input from the beginning and one "backward" recurrent unit that processes the input from the end token. These two units produce a unique output by weighting forward and backward prediction (with learnt weights).

deeplearning.ai

Recurrent Neural Networks

Bidirectional RNN

$$\hat{y}^{<i>} = W_{y_F} \vec{y}_F + W_{y_B} \vec{y}_B$$

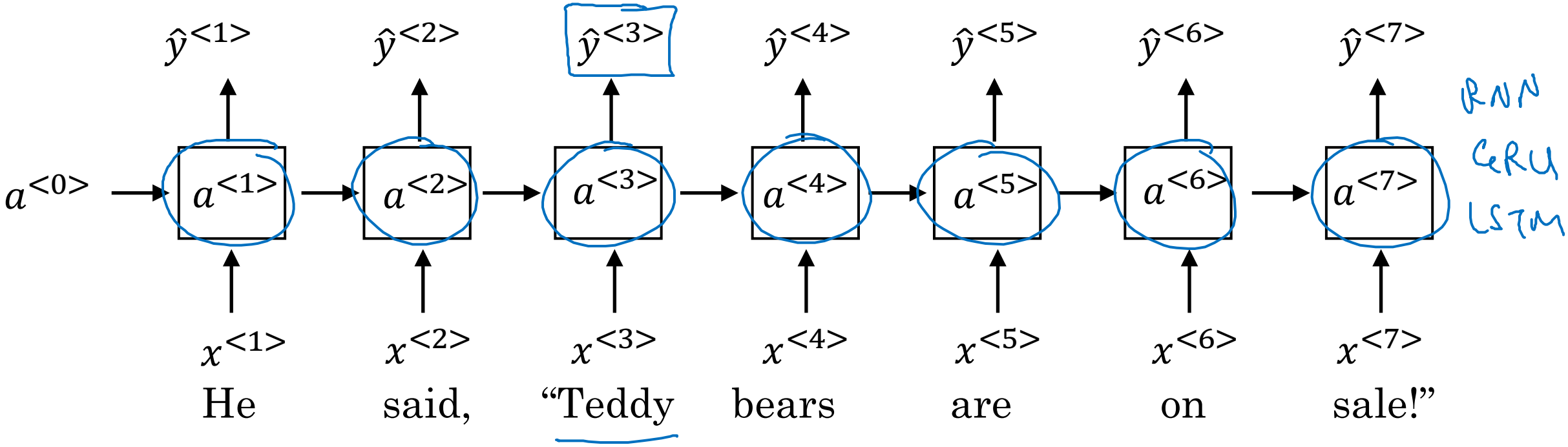


Getting information from the future

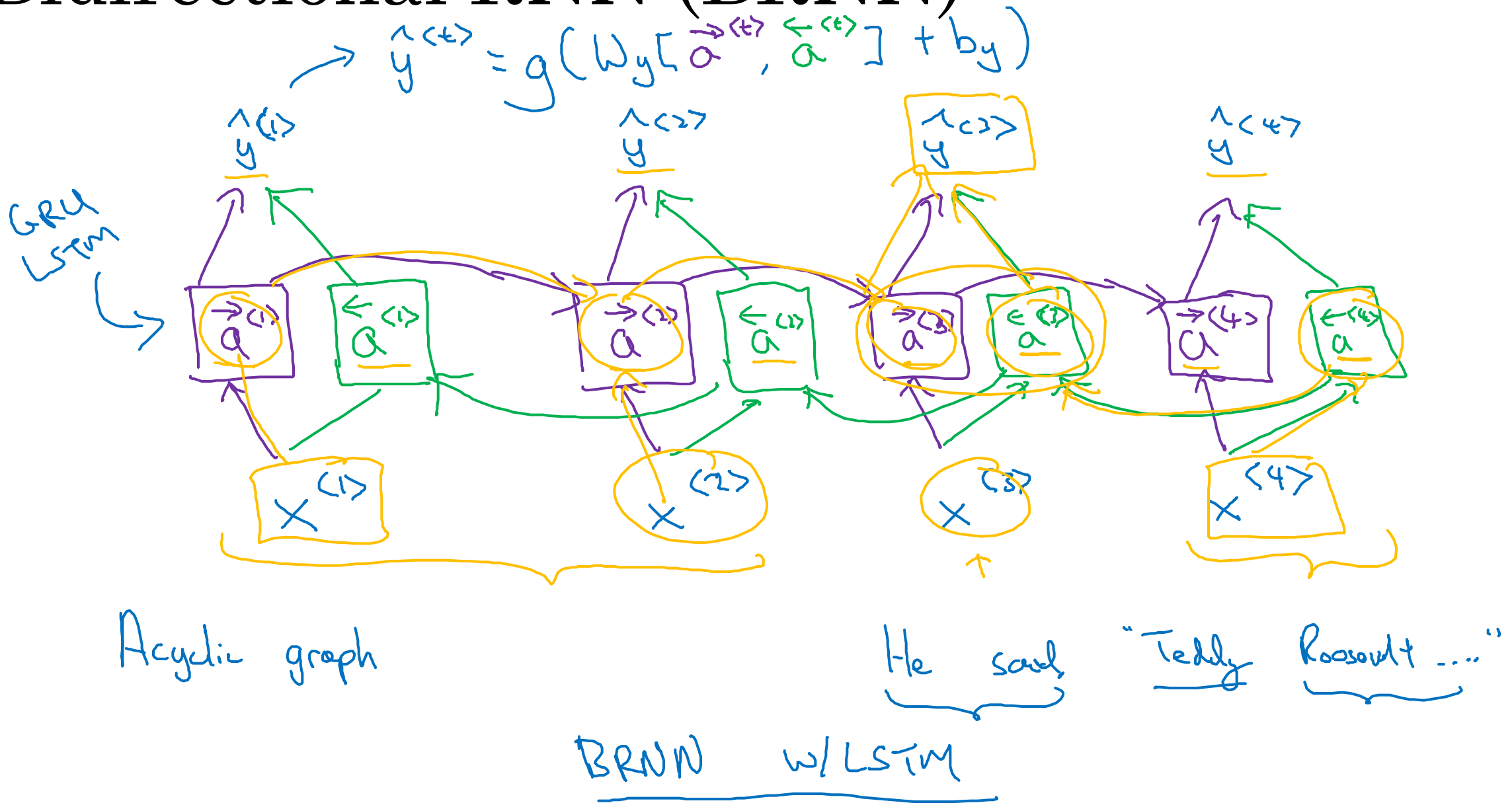
He said, “Teddy bears are on sale!”

He said, “Teddy Roosevelt was a great President!”

Application = named entity recognition



Bidirectional RNN (BRNN)



Obtained stacking recurrent units (RU, GRU, LSTM units).

- "Vertical" stacking from bottom to top; usual "horizontal" time connection.
- Similarly to before, each unit has 2 inputs. Input units like $a^{[1]<i>}$ have inputs $x^{<i>}$ and $a^{[1]<i-1>}$

Recurrent Neural Networks

Hidden units like $a^{[2]<i>}$ have inputs $a^{[1]<i>}$ and $a^{[2]<i-1>}$.

deeplearning.ai

Deep RNNs

It's unusual to find very deep RNNs, usually 3 layers at most. One possibility is to stack few recurrent layers and several standard units.

Deep RNN example