Week 1 Recurrent Neural Networks

Wednesday, 29 August, 2018 21:54

Why sequence models

Wednesday, 29 August, 2018 23:36

Goal: Get an overview of when Recurrent Neural Nets are used.

Recurrent Neural Networks are widely deployed in:

- speech recognition
- natural language processing
- other (time series related tasks)

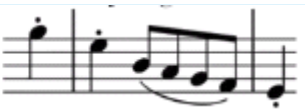
Examples:

x y

Speech recognition  → "The quick brown fox jumped over the lazy dog."

audio clip sequence of words

Screen clipping taken: 2018-08-29 23:36

Music generation ∅ → 

no input (maybe a few notes to provide a pattern) sequence of notes.

Screen clipping taken: 2018-08-29 23:39

Sentiment classification "There is nothing to like in this movie." → ★☆☆☆☆

sequence of words # of stars

Screen clipping taken: 2018-08-29 23:42

DNA sequence analysis AGCCCCTGTGAGGAACTAG → AGCCCCTGTGAGGAACTAG

DNA sequence labeled DNA sequence corresponding to a protein.

Screen clipping taken: 2018-08-29 23:43

Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
	sequence of words		sequence of words

Screen clipping taken: 2018-08-29 23:44

Video activity recognition		→	Running
	Sequence of video frames		activity

Screen clipping taken: 2018-08-29 23:45

Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, Harry Potter met Hermione Granger.
	sequence of words		label the people

Screen clipping taken: 2018-08-29 23:46

Common theme: all these problems involve different types of sequences as input or output.

→ sometimes both input (X) & output (Y) are sequences.

→ sometimes X & Y are sequences of different lengths

→ sometimes only X or Y are sequences.

Notation

Wednesday, 29 August, 2018 23:52

Goal: Learn the notation used for sequence related deep learning.

MOTIVATING EXAMPLES

Problem: Name Entity recognition

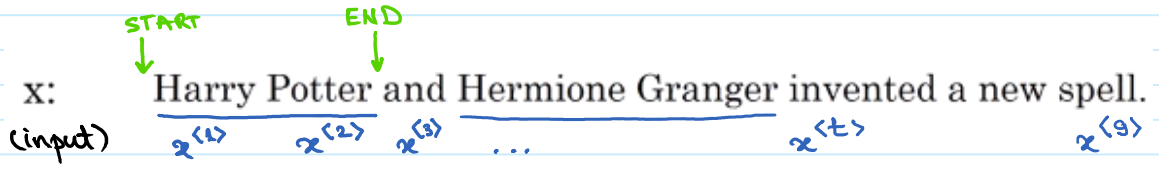
→ Used by search engines to index all people mentioned in news articles in the the last 24 hours.

→ can also be used to find: company names, time, locations, country names currency names, etc in different bodies of text.

Identify the people's name in the Input sentence

.

- FOR A SINGLE TRAINING EXAMPLE (x, y)



Screen clipping taken: 2018-08-29 23:55

$T_x = 9$

y : 1 1 0 1 1 0 0 0 0

(output) $y^{\langle 1\rangle}$ $y^{\langle 2\rangle}$ ... $y^{\langle 9\rangle}$

$T_y = 9$

model outputs a 0 or 1 flag per word that indicates if the word is part of a name (more sofisticated output can be where the START & END of the full name is)

Input 9 words : indexed using angle brackets $x^{\langle t \rangle}$ (t indicates a temporal seq.)

Output 9 features to represent each word : indexed using $y^{\langle t \rangle}$

Indicate the length of the input & output by using T_x and T_y respectively.

→ T_x not always equals T_y

- FOR TRAINING SETS (X, Y):

$x^{(i)\langle t \rangle}$ or $y^{(i)\langle t \rangle}$ denotes the t^{th} component of the i^{th} training example input or output

$T_x^{(i)}$ or $T_y^{(i)}$ indicates the number of elements in the i^{th} training example sequence input or output

REPRESENTING WORDS

As we start exploring Natural Language Processing problems we need to decide how to represent individual words in a sequence.

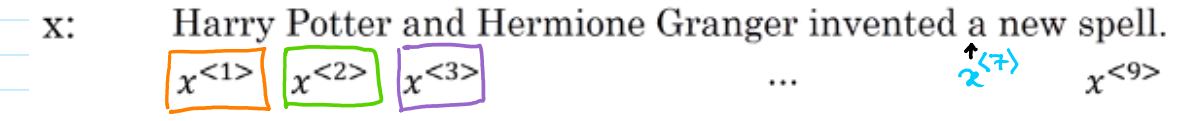
→ what should $x^{\langle 1 \rangle}$ be.

To represent a word in a sequence (i.e input sentence) we'll create a vocabulary (a list of the words we'll use in our representation)

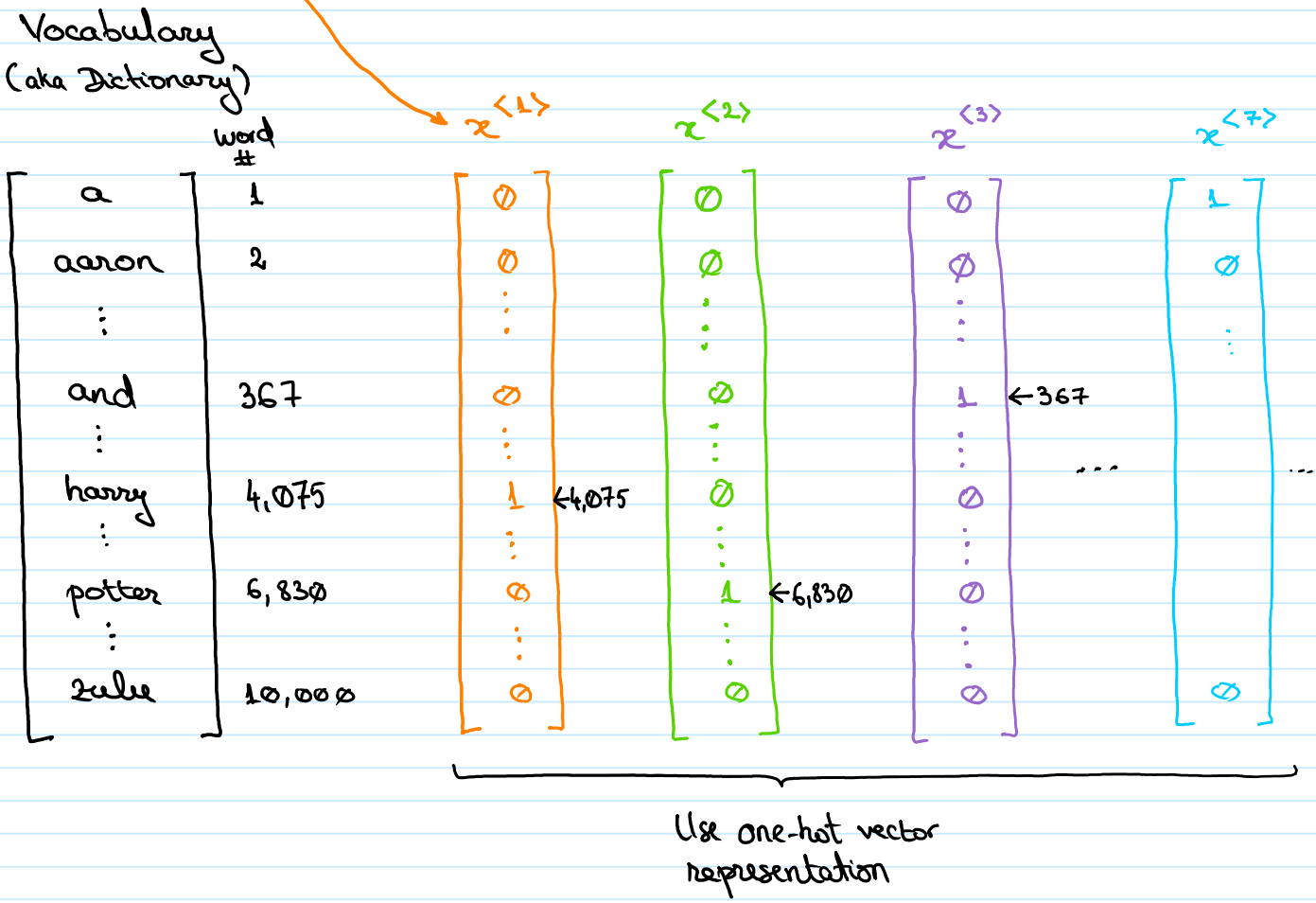
To build a vocabulary :

- scan the training set & choose the top 10k most used words

- use an online dictionary and choose the most common 10k words of the English language.



Screen clipping taken: 2018-08-30 00:42



- In the example above we are using a dictionary of 10,000 words
  → small by today's NLP standards – that use 30k to 50k even 100k to 1M words

- So given a one hot vector $x^{<t>}$ we want to learn a mapping using a sequence classifer (model) that outputs $y^{<t>}$

- For one training example $x^{(i)} \rightarrow y^{(i)}$

$$x = \begin{bmatrix} | & | & & | \\ x^{\langle 1 \rangle} & x^{\langle 2 \rangle} & \cdots & x^{\langle 9 \rangle} \\ | & | & & | \end{bmatrix} \rightarrow y = \begin{bmatrix} y^{\langle 1 \rangle} & y^{\langle 2 \rangle} & \cdots & y^{\langle 9 \rangle} \end{bmatrix}$$

one hot matrix

- What if we encounter a word that is not in the Dictionary?
  - → create a fake word UNK (unknown) to represent words not in the vocabulary.
  - → more to come in later lectures.

$$\begin{bmatrix} a \\ aaron \\ \vdots \\ and \\ \vdots \\ harry \\ \vdots \\ potter \\ \vdots \\ zulu \\ UNK \end{bmatrix}$$

! Recurrent Neural Network Model

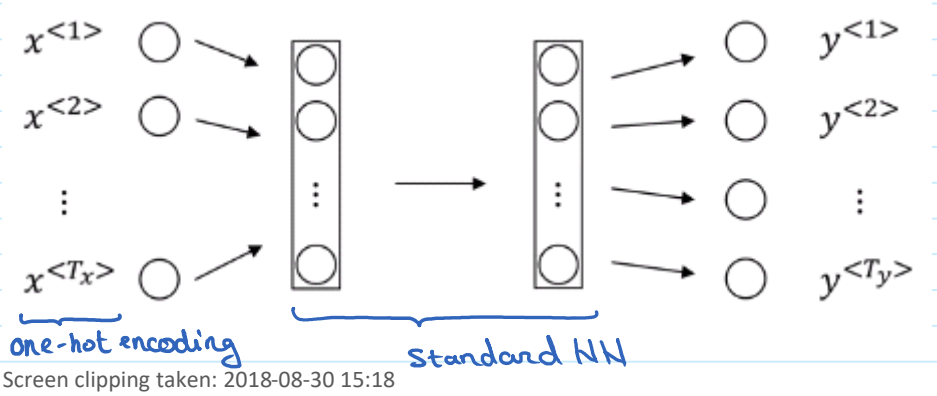
Thursday, 30 August, 2018 15:15

Goal: Learn to build a RNN model to learn the mapping from $x \rightarrow y$.

WHY NOT USE A STANDARD NN?

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

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



Screen clipping taken: 2018-08-30 15:18

In the example above:

- $T_x = 9$ (9 words in the sentence)
- $x^{\langle 1 \rangle} \dots x^{\langle 9 \rangle}$ are 10K elements long one-hot encoded vectors of the dictionary we used.
- $y^{\langle 1 \rangle} \dots y^{\langle 9 \rangle}$ are integers from the set $\{1, 0\}$ indicating whether word $x^{\langle 1 \rangle} \dots x^{\langle 9 \rangle}$ respectively are part of a person's name.

PROBLEMS WITH STANDARD NN:

1) Inputs & Outputs can have different lengths (T_x or T_y) in different examples.

→ we could pad all examples so they match a maximum length example - but it doesn't feel like a good representation.

2) A simple neural net (like above) doesn't share features learned accross different positions of text.

i.e. NN learns the feature (pattern) that the word "Harry" appearing in position $x^{<1>}$ indicates it is part of a name It would be nice to automatically determine that the word "Harry" appearing at position $x^{<t>}$ is also a part of a name.

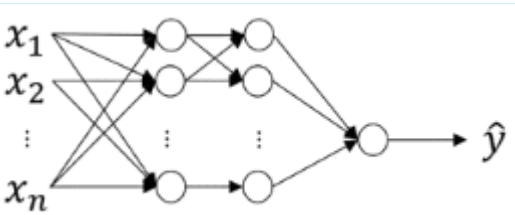
→ This is similar to how a Conv Net uses convolution to generalize learning features in one part of the image to generalize to other parts of the image.

3) similarly to Conv Nets using a better representation will reduce the number of parameters used.

→ in our example our input features are a tensor of size:

(Dictionary Size) × (# of Words in sentence $\leftrightarrow T_x$)

10k × 9 = 90k elements.



REMEMBER: $a_1^{[l-1]}$, $a_2^{[l-1]}$, $a_3^{[l-1]}$ → $a^{[l]} := \sigma(w^T a^{[l-1]} + b)$ → $a^{[l]}$

$(a^{[l]}, a^{[l-1]})$

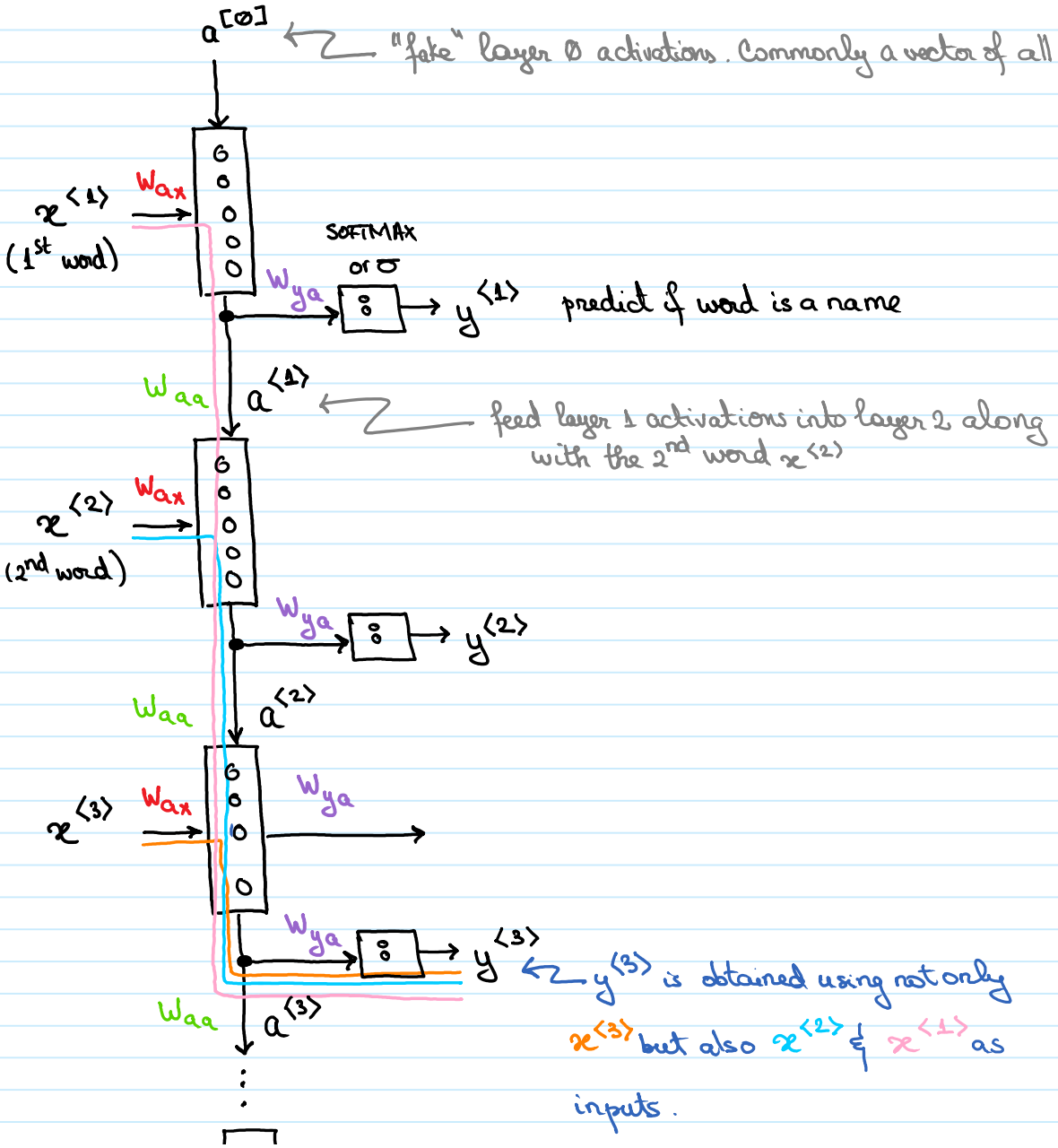
→ so we'll have on the order of 30k × (size of 1st layer) parameters in the first layer (can easily get to 1M parameters)

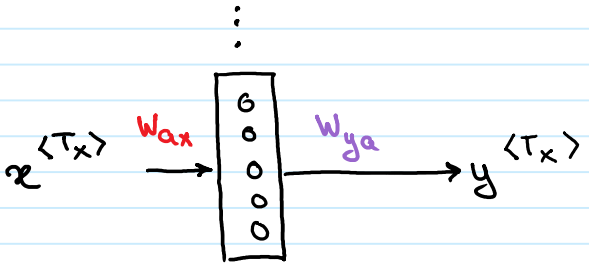
This is not feasible.

To address these problems we'll use Recurrent Neural Networks

RECURRENT NEURAL NETWORKS

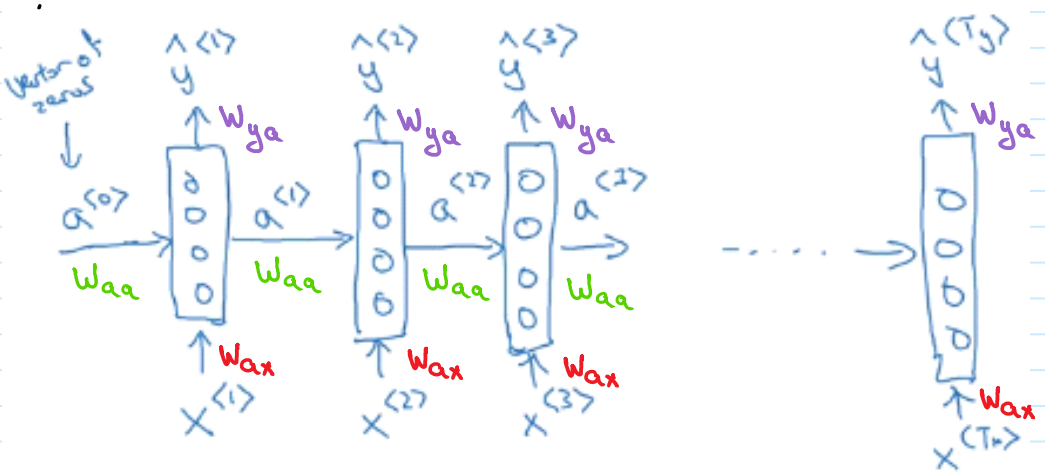
Reading a sentence from left to right we feed each word into a network.





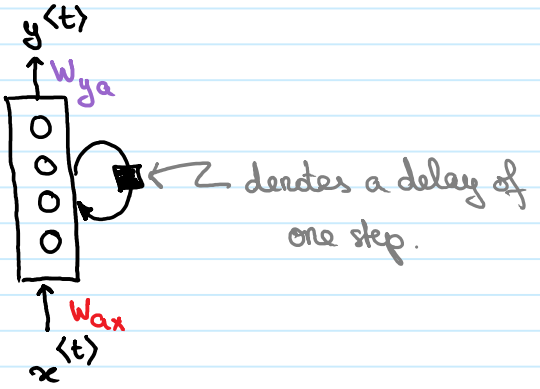
In this case we assume $T_x = T_y$. The architecture changes somewhat if $T_x \neq T_y$

Andrew Ng represented the RNN diagram as below:



Screen clipping taken: 2018-08-30 16:34

In some books or research papers RNNs are drawn as:



The RNN scans through the data from left to right sharing the same parameters (W_{ax}, W_{aa}, W_{ya}) accross all layers.

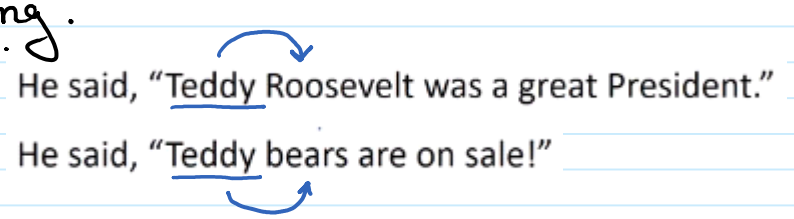
- W_{ax} are a set of parameters that govern the connection from $x^{<t>}$ to the hidden layer t
- The activations are similarly governed by a set of parameters (W_{aa}) that are the same for all layers
- And similarly a set of parameters (W_{ya}) govern the outputs $y^{<t>}$

So in the RNN architecture above information flows only in a single direction (unidirectional RNN).

→ to get a prediction for $y^{<3>}$ we don't only use the information from input $x^{<3>}$ but also information from $x^{<2>}$ and $x^{<1>}$

→ we use information from PREVIOUS words but not from FOLLOWING words

This is a problem as sometimes we would like to use information from words FOLLOWING the current word to determine the current word's meaning.



Screen clipping taken: 2018-08-30 17:17

Ideally we would be able to use information from both earlier & later in the sequence to determine the current output (see Bidirectional RNNs)

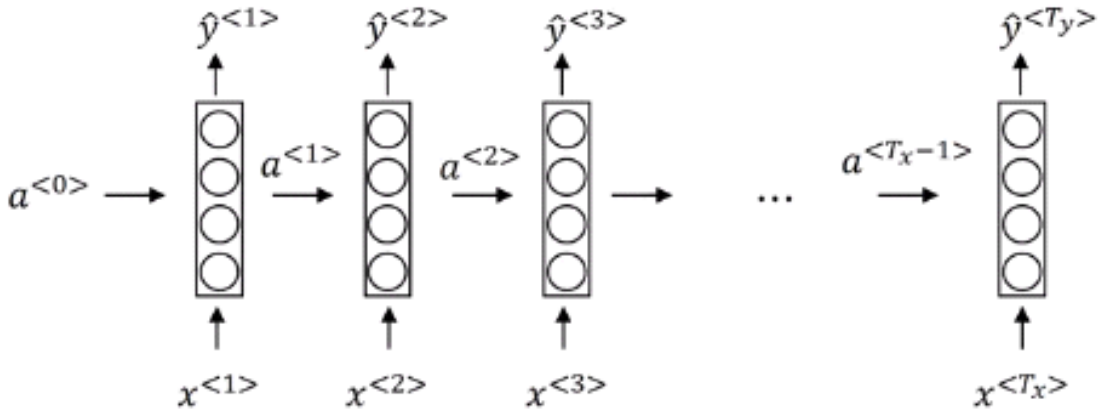
RNNs FORWARD PROPAGATION

NOTE: $a^{<1>} = W_{ax} \cdot x^{<1>}$ $\qquad y^{<1>} = W_{ya} \cdot a^{<1>}$

NOTE: $a^{\langle 1 \rangle} = W_{ax} \cdot x^{\langle 1 \rangle}$ $\quad y^{\langle 1 \rangle} = W_{ya} \cdot a^{\langle 1 \rangle}$

- W_{ax}: refers to what is computed
- $x^{\langle 1 \rangle}$: refers to what w is multiplied by

Given the Unidirectional RNN:



Screen clipping taken: 2018-08-30 17:25

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

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

← most common activation func g is tanh (sometimes ReLU)

$$y^{\langle 1 \rangle} = g_2(W_{ay} a^{\langle 1 \rangle} + b_y)$$

← if we use binary classification we use a sigmoid activation func (or SOFTMAX if we have multiple classes)

$$a^{\langle t \rangle} = g_1(W_{aa} a^{\langle t-1 \rangle} + W_{ax} x^{\langle t \rangle} + b_a)$$

$$y^{\langle t \rangle} = g_2(W_{ya} a^{\langle t \rangle} + b_y)$$

These equations define Forward Prop.

Compressed RNN Notation

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

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

* $W_y = W_{ya}$

Screen clipping taken: 2018-08-30 17:47

W_a denotes W_{aa} and W_{ax} stacked horizontally $\Rightarrow W_a = [W_{aa} \vdots W_{ax}]$

So if: $a^{<t>}$ (100) $= g($ W_{aa} (100, 100) $a^{<t-1>}$ (100) $+$ W_{ax} (100, 10k) $x^{<t>}$ (10k) $+ b_a)$

- $a^{<t-1>}$ & $a^{<t>}$ have 100 elements $\Rightarrow W_{aa}$ has (100, 100) elements.
- $x^{<t>}$ has 10k elements $\Rightarrow W_{ax}$ has (100, 10k) elements.

and

W_a = (100 ↕) $[W_{aa} \vdots W_{ax}]$ has (100, 10100) elements.

(100 → ← 10k →)

Let us use the notation $[a^{<t-1>}, x^{<t>}]$ to denote vectors stacked vertically

$$[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ \vdots \\ x^{<t>} \\ \vdots \end{bmatrix} \begin{matrix} \} 100 \\ \\ \} 10k \end{matrix} \} 10100$$

So $W_a \cdot [a^{\langle t-1 \rangle}, x^{\langle t \rangle}] = \underset{100}{\left[W_{aa} \quad W_{ax} \right]} \cdot \underset{10100}{\begin{bmatrix} a^{\langle t-1 \rangle} \\ x^{\langle t \rangle} \end{bmatrix}}$

$$= W_{aa} \cdot a^{\langle t-1 \rangle} + W_{ax} x^{\langle t \rangle}$$

The advantage of the compressed notation is that we carry around only one parameter matrix (W_a) instead of two (W_{aa} & W_{ax}) which simplifies the notation for when we develop more complex models.

So W_a indicates we are calculating an activation a.

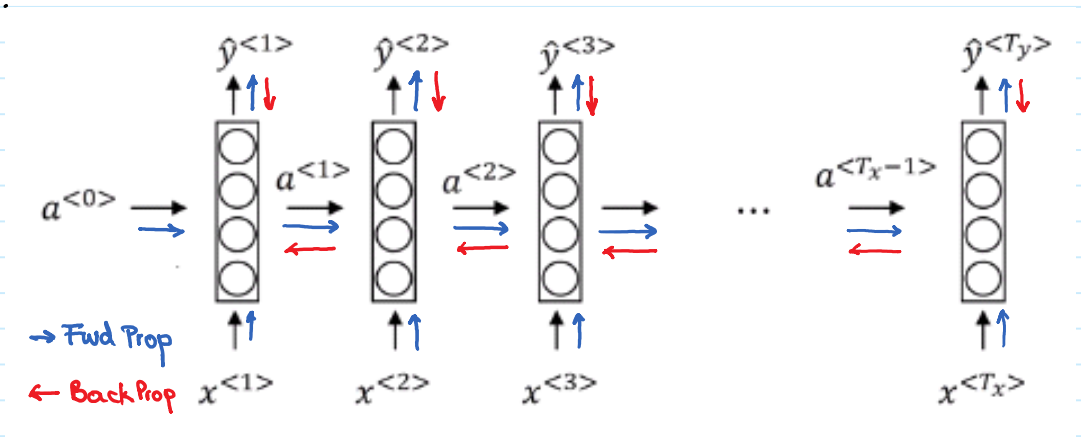
W_y ———— " ———— output y.

Backpropagation through time

Thursday, 30 August, 2018 18:23

Goal: Learn how Back Propagation works in a RNN.

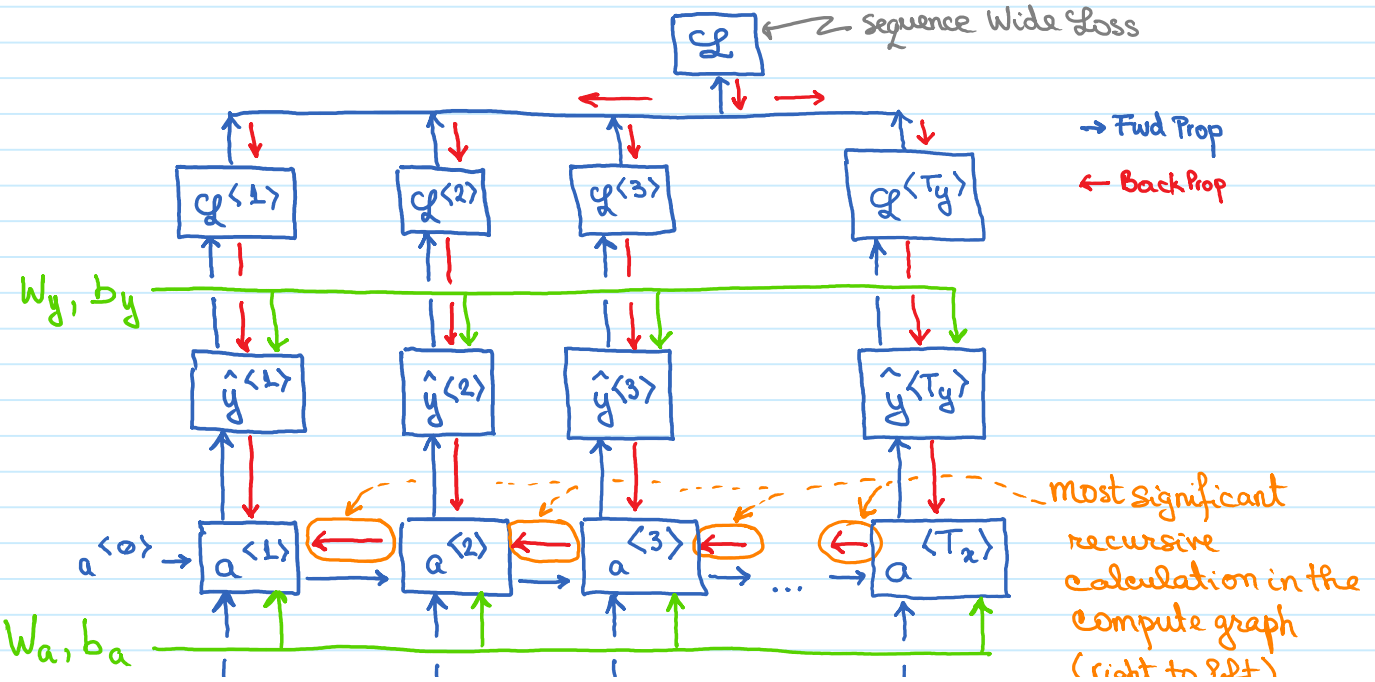
NOTE: Most of the Deep Learning frameworks will compute Back Prop automatically.

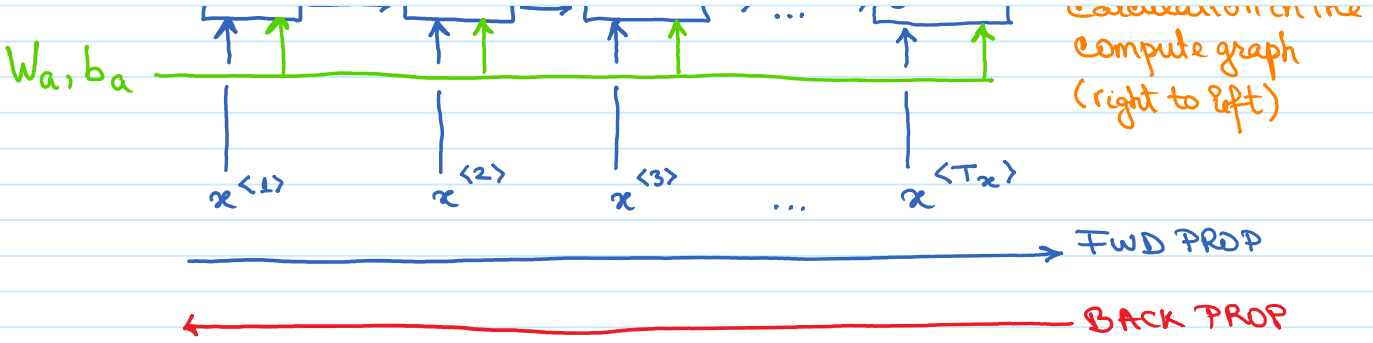


Screen clipping taken: 2018-08-30 18:38

FORWARD & BACK PROPAGATION

To calculate the back prop functions we will build the computation graph below:





For FWD Prop general direction through the graph is from left to right

→ increasing t ↔ foward in time.

For BACK Prop general direction through compute graph is from right to left

→ decreasing t ↔ backward in time ⇒

⇒ Algorithm Recursive Network name is:

BACKPROPAGATION THROUGH TIME

To run backpropagation we need to define a loss function:

- Element wise loss: (for a single word) for word t in the training sequence we determine it is a person's name (ground truth $y^{<t>} = 1$) and our classifier outputs $\hat{y}^{<t>} = 0.1$.

We will use the standard Logistic Regression loss (cross entropy loss) to calculate the loss function:

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

- Sequence wide loss (for all the elements/words in the sequence/sentence)

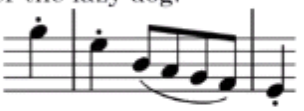
$$\mathcal{L}(\hat{y}, y) = \sum_{t=1}^{T_y} \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

Different types of RNNs

Thursday, 30 August, 2018 22:33

Goal: Learn a wider range of RNN architectures that we can use to tackle a wider range of applications.

When (number of inputs T_x) $\neq$ (number of outputs T_y) we need to use RNN architectures different than Backpropagation through Time.

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

Screen clipping taken: 2018-08-30 22:36

For more information see the blogpost by Andrej Karpathy titled:

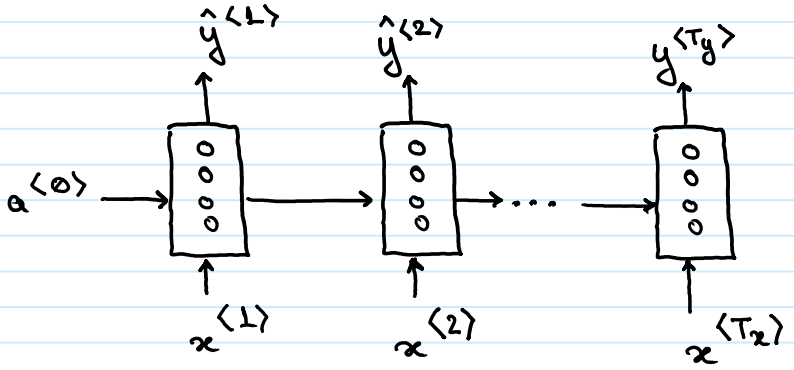
"The unreasonable effectiveness of Recurran Networks."

EXAMPLE OF RNN ARCHITECTURES:

- Backpropagation through time

  Architecture: Many - to - many (many inputs, many outputs)

$T_x = T_y$



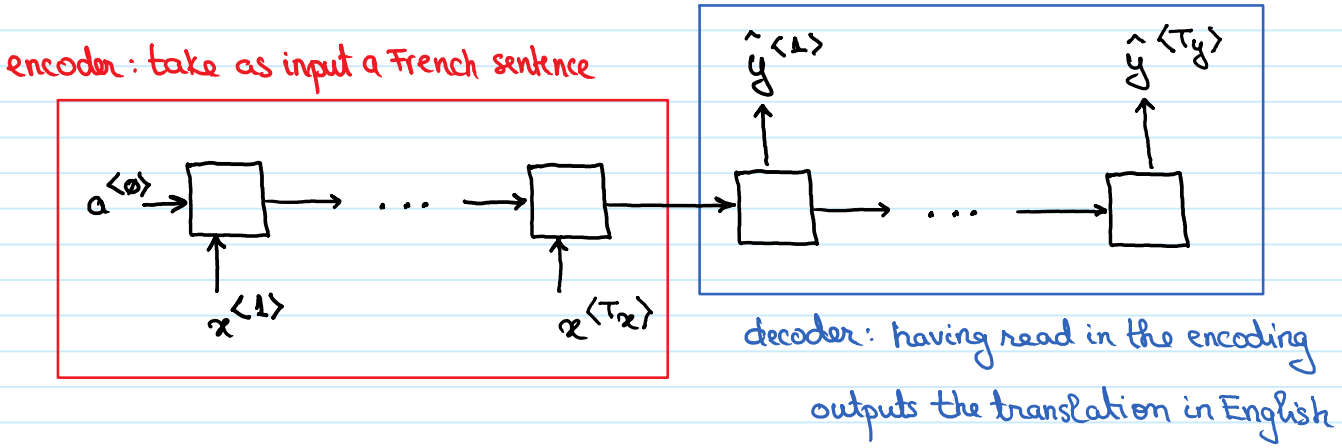
- Machine translation

Architecture: Many - to - many (many inputs, many outputs)

$T_x \neq T_y$ (# of inputs ≠ # of outputs)

A sentence in French doesn't always have the same number of words as its translation in English.

→ First read in the sequence / sentence entirely and only then generate the output



- Sentiment classification

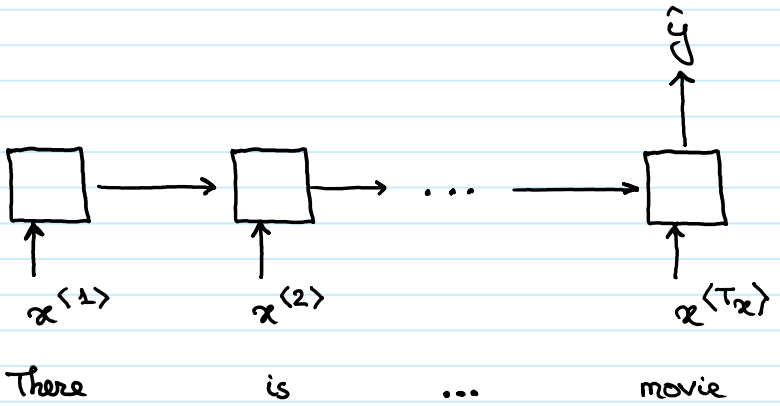
Architecture: Many - to - one (many inputs, one output)

Architecture: Many-to-one (many inputs, one output)

Determine how much a movie was liked from a movie review.

Input x: sequence of words
(i.e. "There is nothing to like in this movie.")

Output y: $\{0, 1\} \leftrightarrow$ {negative, positive}
one ★ to five ★ rating



In this case the RNN reads the entire sequence (sentence) and then outputs a prediction $\hat{y}$ in the last time step (instead of outputting a prediction at every step).

- Standard NN (covered in Course 1 & 2 of this specialization)

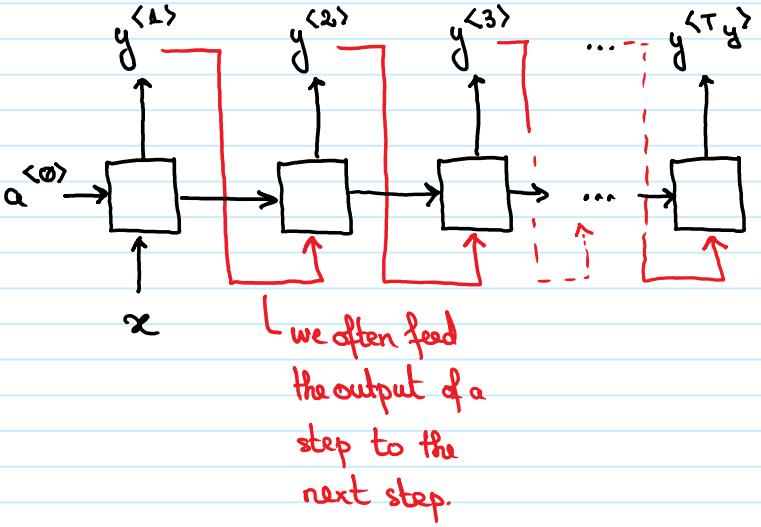
Architecture: One-to-one (one input, one output)

- Music generation

  Architecture: one-to-many (one input, many outputs)

  Input x : an integer indicating what genre of music the output should be
  an integer indicating the first note to be used.
  or
  ϕ / Null / Nothing

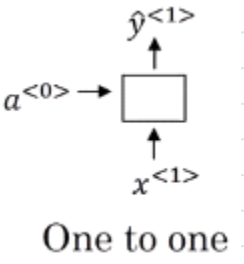
  Output y is set of musical notes



- Attention based architecture

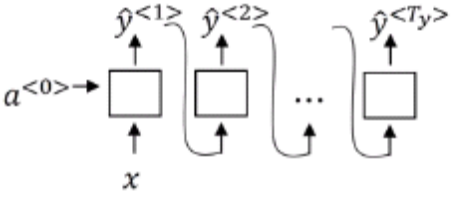
  See last week of this course.

SUMMARY OF RNN ARCHITECTURES

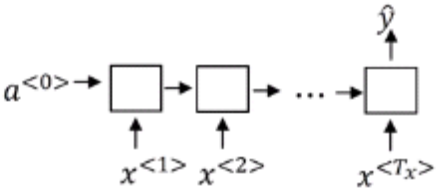


One to one

Screen clipping taken: 2018-08-30 23:55

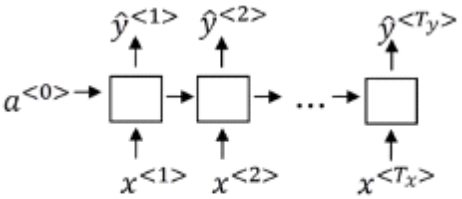


One to many



Many to one

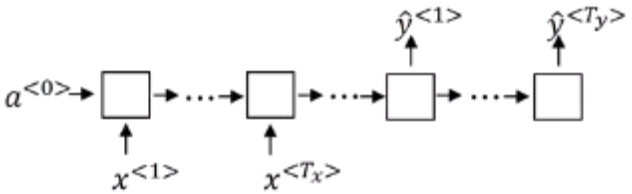
Screen clipping taken: 2018-08-30 23:57



Many to many

$T_x = T_y$

Screen clipping taken: 2018-08-30 23:57



$T_x \neq T_y$

Screen clipping taken: 2018-08-30 23:58

Language model and sequence generation

Friday, 31 August, 2018 00:00

Goal: Learn some of the subtleties of sequence generation tasks (like language modeling → a subdomain of natural language processing)

WHAT IS LANGUAGE MODELLING?

Let's say we are building a speach recognition system and we hear a sentence that has a homophone word like below:

The apple and pair salad.

The apple and pear salad. → This sentence is more likely.

Screen clipping taken: 2018-08-31 00:16

A speach recognition system uses a language model to tell it what the probability of encountering either of the two sentences is.

$P(\text{The apple and pair salad}) = 3.2 \times 10^{-13}$

$P(\text{The apple and pear salad}) = 5.7 \times 10^{-10}$ ← more likely by a factor of 1,000

Screen clipping taken: 2018-08-31 00:22

$P(\text{sentence}) = ?$

So a language model:

- takes as input a sentence

outputs a probability of encountering the sentence in the corpus of spoken sentences for that language.

→ or maybe the probability of that sentence being generated in that language.

→ or maybe the probability of that sentence being generated in that language.

Input: $y^{\langle 1 \rangle}, y^{\langle 2 \rangle}, \dots y^{\langle T_y \rangle}$

(for a language model it's more useful to represent the sentence as outputs y instead of inputs x)

Output: $P(y^{\langle 1 \rangle}, y^{\langle 2 \rangle} \dots y^{\langle T_y \rangle})$ probability of this particular sequence of words.

Language models are used both for

- SPEECH RECOGNITION
- MACHINE TRANSLATION (output only sentences that are likely)

How to Build a Language Model with an RNN?

We need a training set: a large corpus of English text
(a large set of texts in the language of interest)

Input sentence:

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

$y^{\langle 1 \rangle}$ $y^{\langle 2 \rangle}$ $y^{\langle 3 \rangle}$ $y^{\langle 9 \rangle}$

Screen clipping taken: 2018-08-31 00:38

- Tokenize the sentence:

  For example using a 10,000 words vocabulary (see Notation) map each of the words in the input sentences to one-hot vectors or indices in the vocabulary.

We might also want to model when sentences end.

So we could add an extra token <EOS> to mark this (end of sentence)

Note: <EOS> is not used in the programming assignment.

We can also determine if the punctuation is tokenized or ignored in this step.

Remember that the vocabulary uses the <UNK> token to indicate words that are not contained within it. (UNKnown)

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

<UNK>

Screen clipping taken: 2018-08-31 00:49

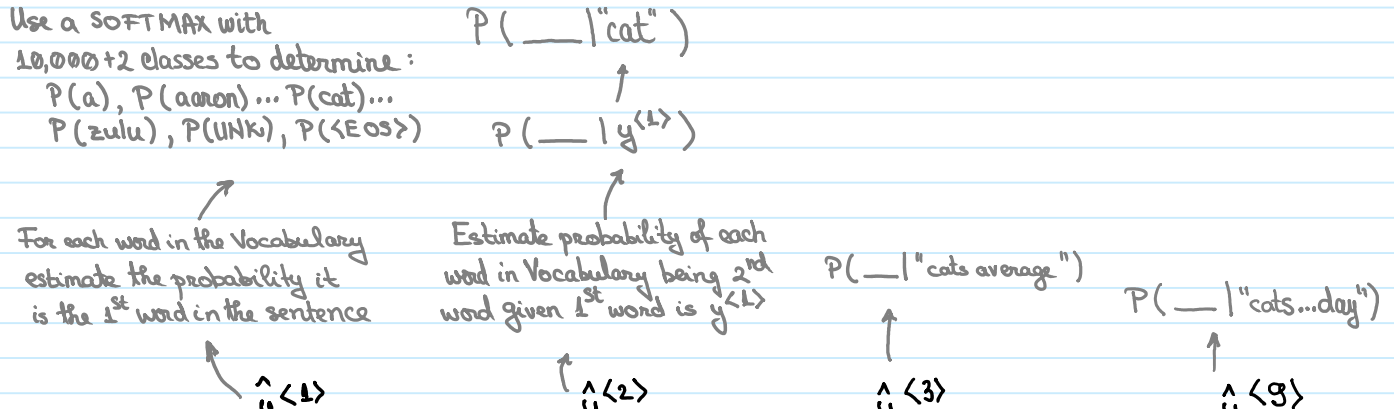
- Build an RNN (it models the probability of the input sequence)

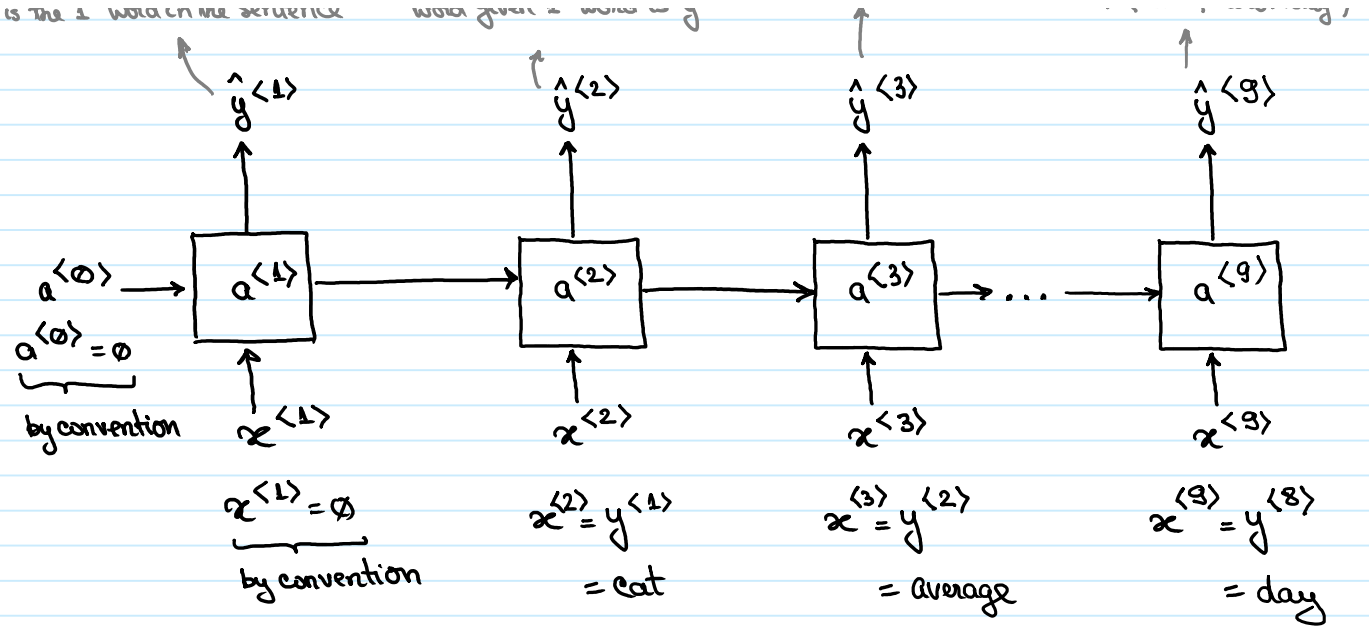
Note: we'll end up setting $x^{\langle t \rangle} = y^{\langle t-1 \rangle}$

We continue using the sentence below as input:

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

Screen clipping taken: 2018-08-31 01:17





- Hopefully (once trained) the output of all the steps in the RNN predicts the training examples sentences (word sequences) with high probability.
- The RNN learns to predict one word at a time (going from left to right)

■ What is the Cost Function used to train this RNN?

For each element t of the sequence

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

Overall loss (for entire sentence):

$$\mathcal{L}(y) = \sum_t \mathcal{L}^{\langle t\rangle}(\hat{y}^{\langle t\rangle}, y^{\langle t\rangle})$$

Once trained the RNN will be able to predict given a set of previous words what is the probability of a word being the next word in a sentence

P(tractor | "cat average 15")

P(hours | "cat average 15")

Given a new sentence of 3 words $y = (y^{<1>}, y^{<2>}, y^{<3>})$ the probability of the entire sentence is:

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

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

Sampling novel sequences

Friday, 31 August, 2018 16:51

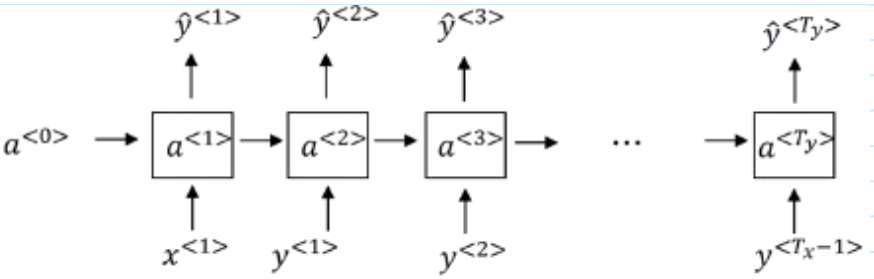
Goal: Learn how to assess the performance of a sequence model (i.e. language model) by getting it to sample new sequences.

SAMPLING A SEQUENCE FROM A TRAINED RNN

A sequence model estimates the probability of any given sequence exists based on the training data used to train it

$$P(y^{\langle 1\rangle}, y^{\langle 2\rangle}, y^{\langle 3\rangle} \dots y^{\langle T_y\rangle}) = x\%$$

We train the network using the following architecture:



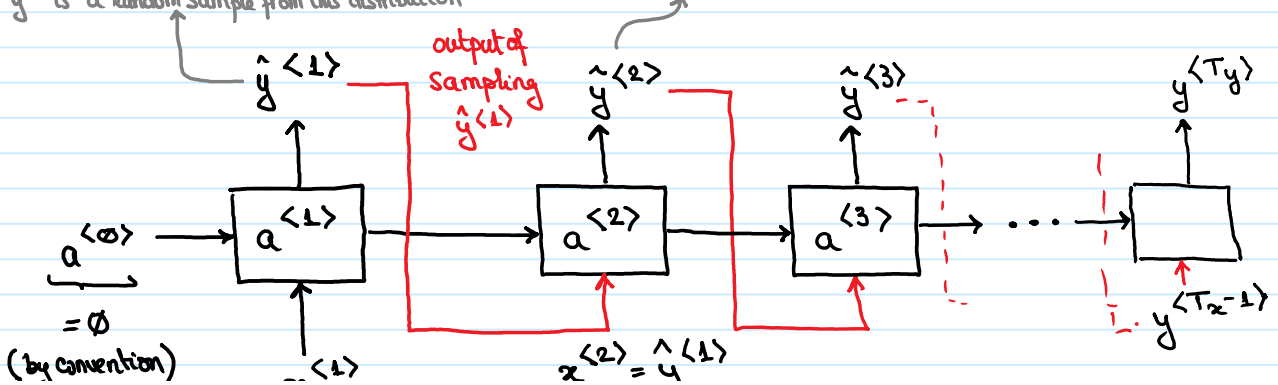
Screen clipping taken: 2018-08-31 16:56

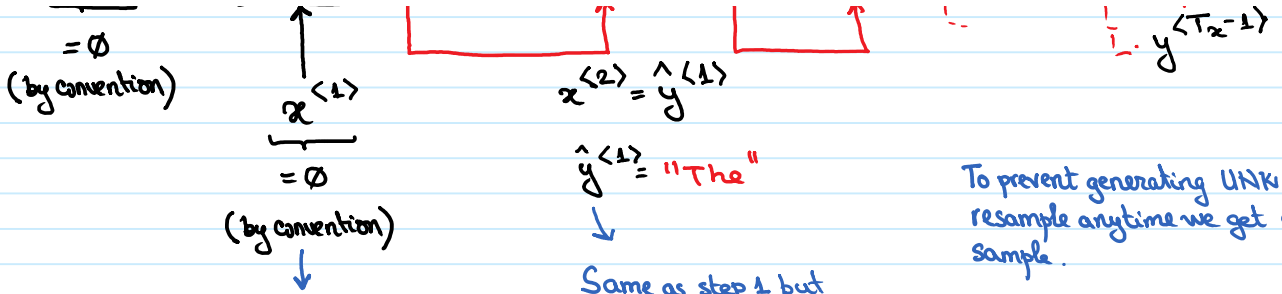
To sample we use a slightly different structure:

- 1st step contains a probability distribution over all words in the vocabulary where each word in the vocabulary is assigned the probability of it being the 1st word of the sequence
- this is represented as SOFTMAX classes
- $\hat{y}^1$ is a random sample from this distribution

$\hat{y}^{\langle 2\rangle}$ is a sample from the distribution $P(\_\_ | \text{"the"})$

Determine the sequence / sentence ended
- when an $\langle EOS\rangle$ is generated (if we use $\langle EOS\rangle$ in Vocabulary) or
- set a predifined length (i.e. 20 steps / words) to sample before stopping





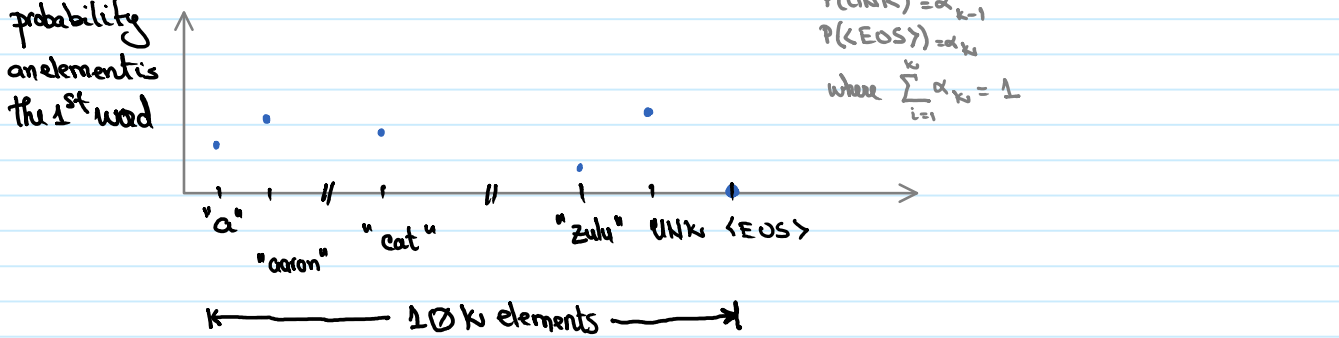
In the 1st step randomly sample the SOFTMAX distribution $\hat{y}^{\langle 1 \rangle}$ and use it as input for 2nd step

- Can sample using: np.random.choice()

Same as step 1 but distribution is conditional on $\hat{y}^{\langle 1 \rangle}$ value.

To prevent generating UNK tokens resample anytime we get a UNK sample.

Step 1 distribution:



CHARACTER-LEVEL LANGUAGE MODEL:

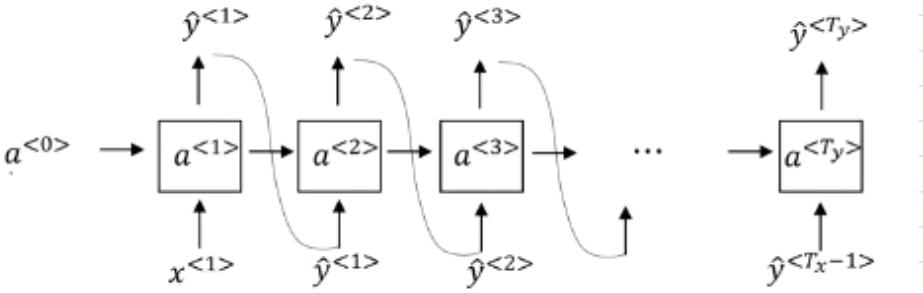
The sequence model above is a word level RNN (i.e. the Vocabulary elements are words from English).

We can also build a character level RNN

→ Vocabulary elements are alphabet characters.

Vocabulary = [a, b, c, ... z, ␣, ., ,, ;, ∅, ..., 9, A, ... Z]

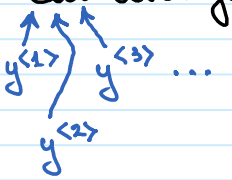
punctuation + space.



Screen clipping taken: 2018-08-31 23:13

So $y^{<1>}$, $y^{<2>}$, $y^{<3>}$ are characters in the training data in this case.

"Cat average 15 hours..."



Character-level model advantages / disadvantages:

PRO

- Don't have to worry about unknown word <UNK> token as any word can be generated from the character Vocabulary

CON

- The sequences used by character-level models are much longer. This rapidly increases computational demands and becomes not feasible.
  English sentences are ~ 10 to 20 words long but can have an order of magnitude more characters.

The trend in Natural Language Processing is to use word-level language models with a few exceptions where character-level models are used.
→ as more compute becomes available the balance might start favoring character-level models.

We can now build language generating RNNs.

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.

Screen clipping taken: 2018-08-31 23:36

Vanishing gradients with RNNs

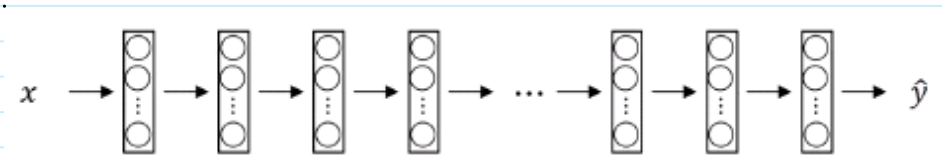
Friday, 31 August, 2018 23:38

Goal: Explore the Vanishing Gradient problem that arises often in RNNs.

RNNs we have seen so far are not good at capturing long-term dependencies like below:

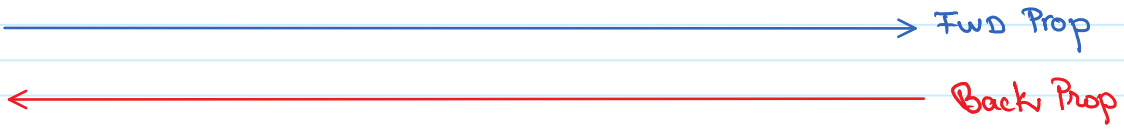
The **cat**, which already ate [...] , **was** full.
The **cats**, which already ate [...] , **were** full.

This is because of the Vanishing Gradient problem:



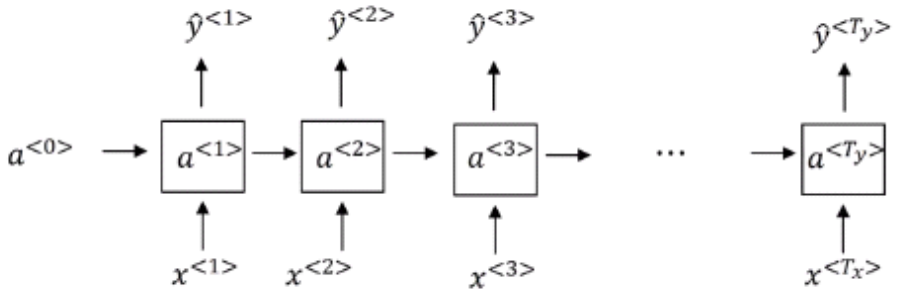
100 layers + deep neural net

Screen clipping taken: 2018-09-01 00:11



The gradient from the deep layers has a very difficult time propagating back to early layers.

An RNN can have many tens / hundreds / thousands of layers as each step is actually a layer. So they exhibit the same problem in that the parameters in early time steps (layers) are barely modified by the gradients from deeper layers.



Screen clipping taken: 2018-08-31 23:40

→ Fwd Prop

← Back Prop

In practice this means it's difficult for the RNN to learn if it saw a singular noun or plural noun (cat vs cats) earlier in the sequence so that later on it uses the "was" or "were" verb.

→ English has an arbitrary number of words in between noun & verb.

The basic RNN model we have been using has many local influences.

- That is an element is influenced much more by close neighbours in the sequence than it is influenced by distant neighbours in the sequence.
- i.e. $y^{<3>}$ is easily influenced by $y^{<2>}$ & $y^{<4>}$ but it is more difficult for the RNN to learn to influence $y^{<3>}$ by $y^{<T_y>}$.

→ the error cannot backpropagate easily.

Note:

In general when training RNNs we encounter Vanishing Gradient problems more often than Exploding Gradient problems.

→ when Exploding Gradient problems do appear they are easier to identify as the parameters start taking the NaN values

(a result of a calculation overflow) and the network stops working.

→ to resolve Exploding Gradients we can use Gradient Clipping (we threshold the gradient values)

REMEMBER:

Vanishing Gradient problem are more common in RNN and are more difficult to detect and resolve.

Gated Recurrent Unit (GRU)

Saturday, 01 September, 2018 00:57

Goal: Learn how using Gated Recurrent Units (a modification to the RNN hidden layer) allows us to deal with Vanishing Gradient problems and thus makes it much easier to capture larger time-dependencies using RNNs.

RNN UNIT

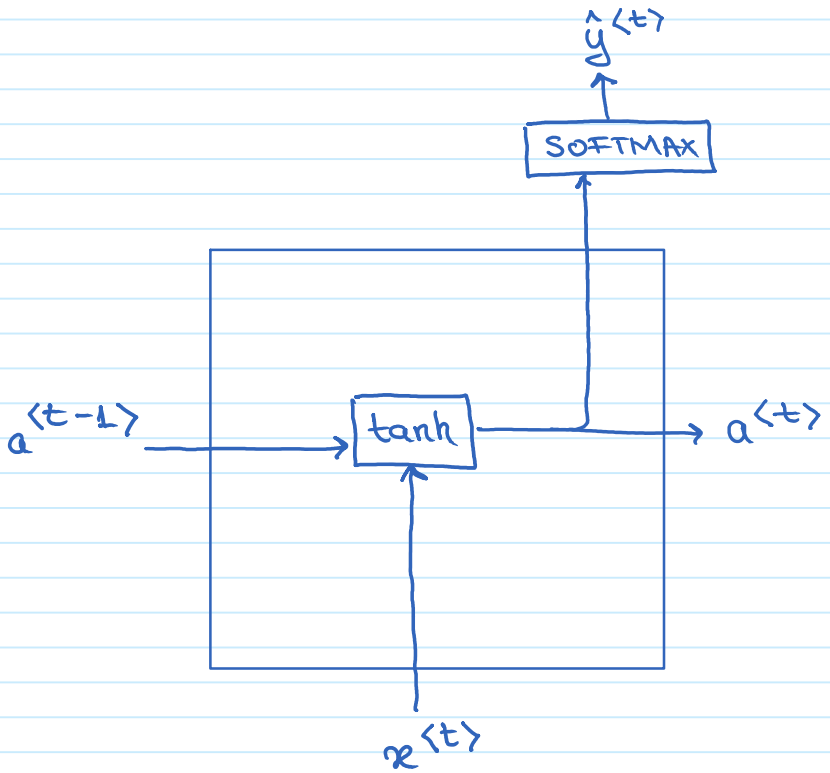
The activation at time (aka step / layer) t of an RNN is:

usually $g() = \tanh$

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

Screen clipping taken: 2018-09-01 01:06

activation from previous time step & the current input.



GATED RECURRENT UNIT (SIMPLIFIED)

GRUs were developed mostly in these two papers:

[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]

Screen clipping taken: 2018-09-01 01:13

Remember in the sentence below the RNN needs to learn that "cat" is singular so that it uses "was" rather than "were".

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

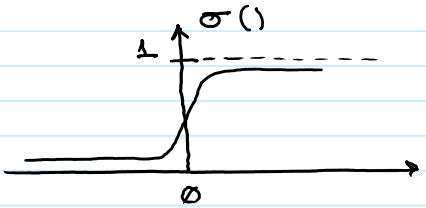
Screen clipping taken: 2018-09-01 01:16

- The GRU unit adds a new unit called C (memory cell) which will remember that the cat is singular or plural.
- At time t the memory cell unit C will have the value $c^{\langle t \rangle}$.
  → actually for RNNs $c^{\langle t \rangle} = a^{\langle t \rangle}$
  (we want to keep the concepts separate because for LSTMs NN $c^{\langle t \rangle} \neq a^{\langle t \rangle}$)
- Every time step (aka layer) we will consider whether to overwrite the memory cell unit value with a new value $\tilde{c}^{\langle t \rangle}$ (candidate for replacing $c^{\langle t \rangle}$)

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

- The important element of a GRU is the update gate (denoted using gamma - Γ_u)

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



$\Gamma_u \rightarrow (0; 1)$

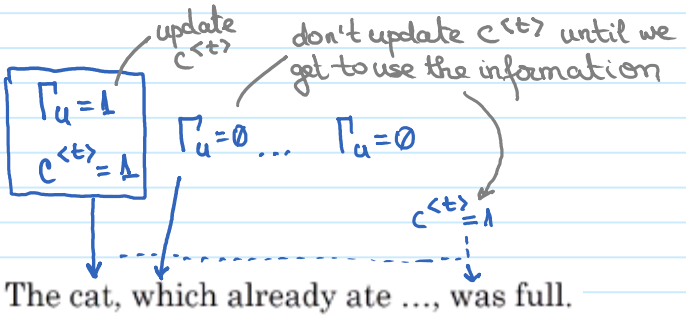
for most values σ is either 0 or 1

Similarly an intuitive way of using

Γ_u is to associate it with either $\emptyset$ or 1.

The update gate decides whether the value of the memory cell gets updated or not.

EXAMPLE:



Maybe $c^{<t>} = 1$ or $\emptyset$ depending on whether the subject of the sentence is singular or plural.

The gate Γ_u determines when we update $c^{<t>}$

→ in this case when we encounter "the cat" update the value of $c^{<t>}$ because we encountered a new concept.

→ after we use it to determine "was full" we no longer need $c^{<t>}$ so we can reset it.

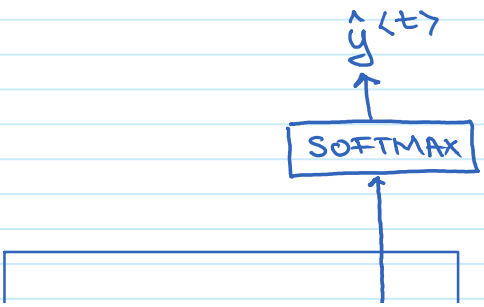
The actual value we will use for the GRU is:

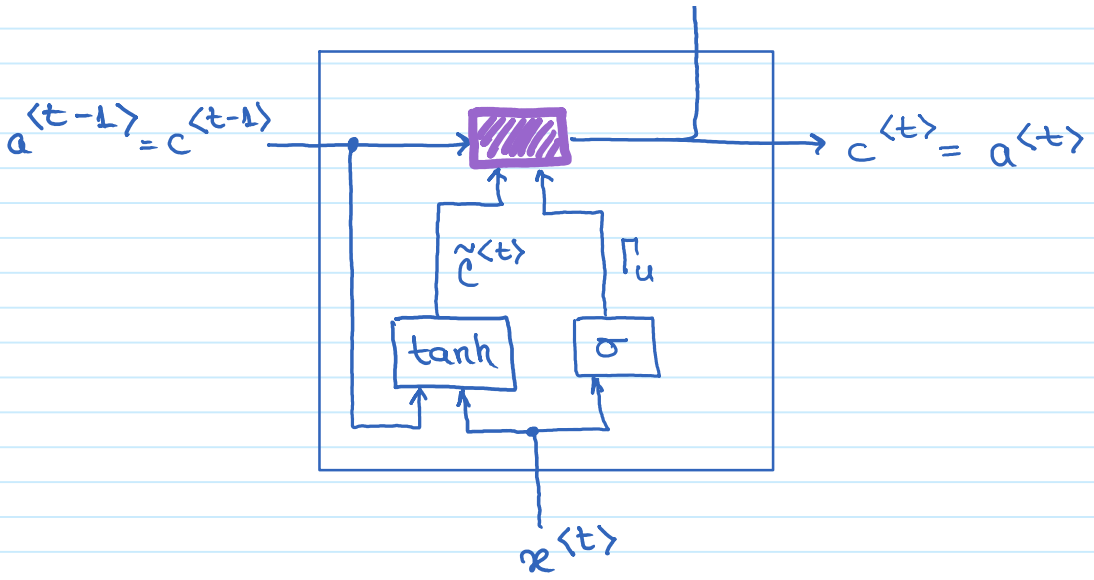
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

so when $\Gamma_u = 1 \Rightarrow c^{<t>} = \tilde{c}^{<t>}$ (UPDATE $c^{<t>}$)

$\Gamma_u = \emptyset \Rightarrow c^{<t>} = c^{<t-1>}$ (DON'T UPDATE $c^{<t>}$)

keep remembering the old value.





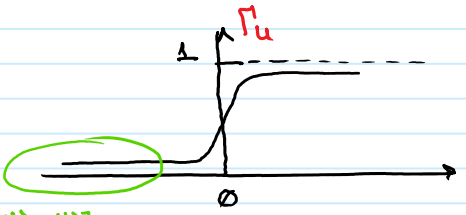
This is the simplified Gated Recurrent Unit.

How does this address the vanishing gradient problem?

- Remember:

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

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



- As long as $W_u\left[c^{\langle t-1\rangle}, x^{\langle t\rangle}\right] + b_u$ is a large negative value it is easy to keep Γ_u to be very small (say 0.00001) and the value of $c^{\langle t\rangle}$ is maintained very closely to $c^{\langle t\rangle} \simeq c^{\langle t-1\rangle}$
- When the gradients (dW_u) are small - we are dealing with a Vanishing Gradient problem — the value of Γ_u does not change much (i.e. it stays close to the initial 0.00001) so the value of $c^{\langle t\rangle}$ can be maintained over many, many steps until the step it is used in.

This allows the network to learn very long range dependencies.

IMPLEMENTATION DETAILS:

- We learned that:

$(100, 1)$ $\boxed{c^{\langle t \rangle}} = a^{\langle t \rangle}$

So if we have a 100 dimensional activation vector then $c^{\langle t \rangle}$ is also 100 dimensional

And also $\tilde{c}^{\langle t \rangle}$ & Γ_u are 100 dimensional vectors

$$(100, 1) \quad \boxed{\tilde{c}^{\langle t \rangle}} = \tanh\left(W_c \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle} \right] + b_c \right)$$

$$(100, 1) \quad \boxed{\Gamma_u} = \sigma\left(W_u \left[c^{\langle t-1 \rangle}, x^{\langle t \rangle} \right] + b_u \right)$$

So the multiplications in the assignment of $c^{\langle t \rangle}$ are element-wise multiplications.

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

element wise multiplication

- Γ_u is a 100 dimensional vector with values that are almost all 0 or 1s (or very close to 0 or 1) that indicate which values of the 100 dimensional memory cell $c^{\langle t \rangle}$ we want to keep and which ones we want to update respectively.
  - → in practice Γ sometimes takes values closer to 0.5 but a good heuristic is to think it takes values of either 0 (maintain $c^{\langle t \rangle}$ value) or 1 (update $c^{\langle t \rangle}$ value)

  → one value in $c^{\langle t \rangle}$ could be associated with singular or plural nouns, another can be used to keep track of food, and so on.

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

$c_1^{\langle t \rangle}$ $c_2^{\langle t \rangle}$

we can individually remember or forget the different "pieces of

information using element wise multiplication by Γ_u.

FULL GRU DETAILS

The full GRU has an additional gate Γ_r used as follows:

update candidate

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

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

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

how relevant is $c^{<t-1>}$ to computing the next update candidate $\tilde{c}^{<t>}$

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

The cat, which ate already, was full.

Screen clipping taken: 2018-09-01 15:52

The full GRU has been tunned over many iterations by many researchers to model long range effect, address Vanishing Gradient problems and be as robust & useful for a large range of problems.

An alternative to GRU architecture is the LSTM.

In the GRU literature sometimes a different notation is used.

$\tilde{c}^{<t>} \rightarrow \tilde{h}$

$\Gamma_u \rightarrow u$

$\Gamma_r \rightarrow r$

$c^{<t>} \rightarrow h$

Long Short Term Memory (LSTM)

Saturday, 01 September, 2018 16:07

Goal: Learn how to use Long Short Term Memory (LSTM) architecture - a more powerfull alternative to the Gated Recurrent Unit (GRU) - that also allows us to learn long range connections in an input sequence.

[Hochreiter & Schmidhuber 1997. Long short-term memory] ← this is a seminal paper that is difficult to understand as it goes into the details of vanishing Gradient problems.

Screen clipping taken: 2018-09-01 16:17

GRU & LSTM

GRU

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

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

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

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

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

Screen clipping taken: 2018-09-01 16:18

LSTM

$a^{<t>} \neq c^{<t>}$

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

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

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

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

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

(element-wise multiplication)

So for $c^{<t>}$ we have the option of keeping the old value & just adding the new value

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

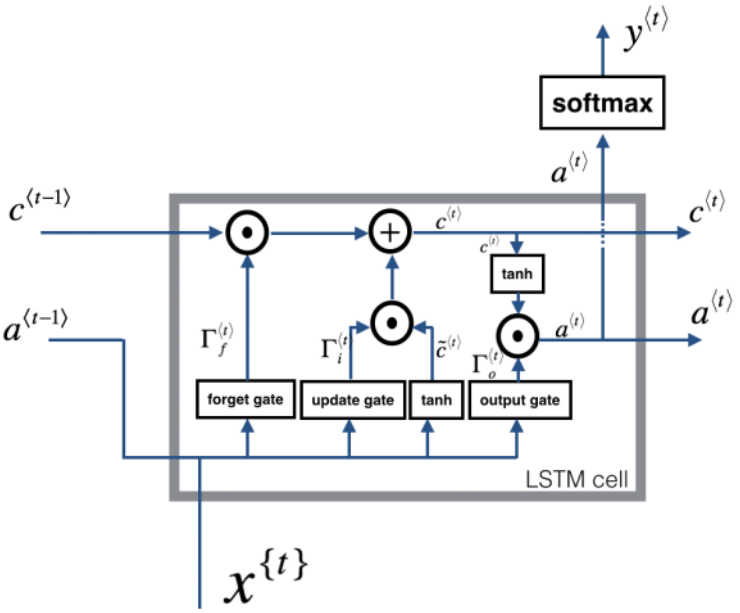
LSTM DIAGRAM

Takeaways:

- all gate values Γ are computed using $a^{<t-1>}, x^{<t>}$

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$
$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$
$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$

Screen clipping taken: 2018-09-01 16:33

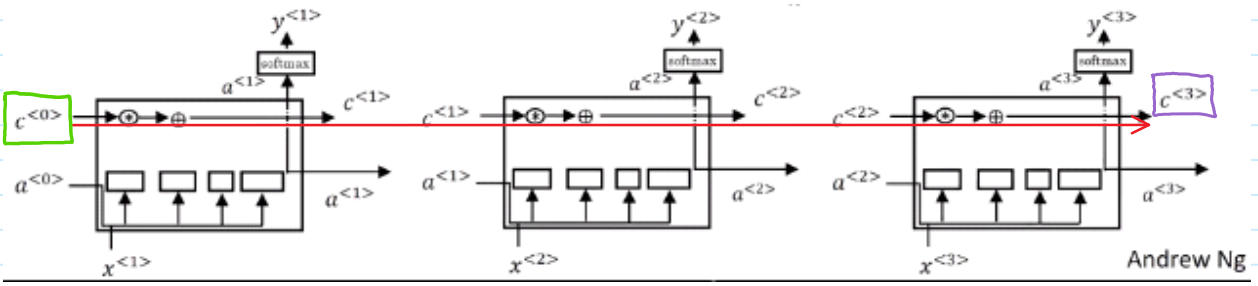


$$\Gamma_f^{\langle t \rangle} = \sigma(W_f[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f)$$
$$\Gamma_u^{\langle t \rangle} = \sigma(W_u[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u)$$
$$\tilde{c}^{\langle t \rangle} = \tanh(W_C[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_C)$$
$$c^{\langle t \rangle} = \Gamma_f^{\langle t \rangle} \circ c^{\langle t-1 \rangle} + \Gamma_u^{\langle t \rangle} \circ \tilde{c}^{\langle t \rangle}$$
$$\Gamma_o^{\langle t \rangle} = \sigma(W_o[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o)$$
$$a^{\langle t \rangle} = \Gamma_o^{\langle t \rangle} \circ \tanh(c^{\langle t \rangle})$$
$$y^{\langle t \rangle} = \text{SOFTMAX}(W_y a^{\langle t \rangle} + b_y)$$

Screen clipping taken: 2018-09-01 16:33

The diagram is inspired by the blogpost titled "Understanding LSTM networks" by Chris Ola

If we daisy-chain a few LSTM units notice that it is relatively easy for a value $c^{<0>}$ to pass unmodified through all the steps all the way to $c^{<3>}$ if Γ_u, Γ_f & Γ_o are set appropriately



Screen clipping taken: 2018-09-01 16:38

This is the reason why LSTM architectures (as well as GRU) are good at memorizing values over many time steps (aka layers).

- There are a few other LSTM variations
  - Peephole connection

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

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

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

in which the gate value depends on the previous memory cell value

NOTE: The memory cells $c^{\langle t\rangle}$ are vectors (i.e. 100 elements) and the gates $\Gamma_u, \Gamma_f, \Gamma_o$ are also vectors where each element in the gate affects only their associated memory cell.

There is no consensus of when to use GRUs vs LSTMs.

→ GRUs are simpler → we can add more gates
→ computationally faster

→ LSTM is more powerful & flexible as it has more gates.

Bidirectional RNN

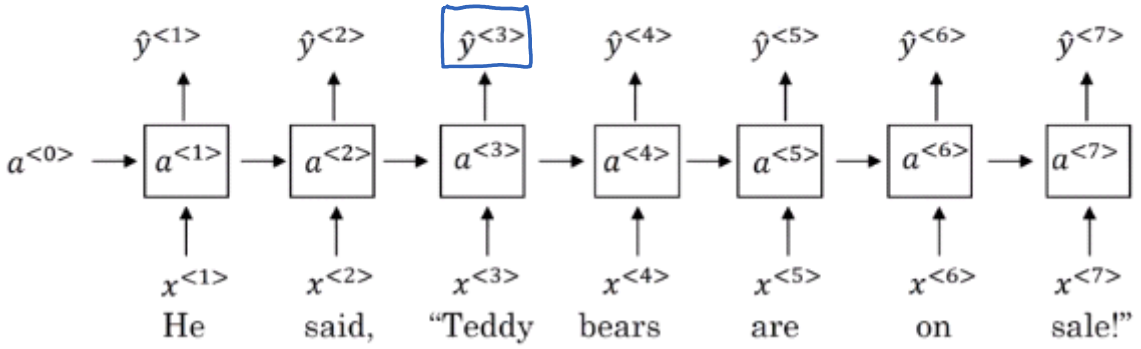
Saturday, 01 September, 2018 16:57

Goal: Learn how to use bidirectional RNN architecture to take information from both previous & later time steps.

GETTING INFORMATION FROM THE FUTURE

He said, "Teddy bears are on sale!"

He said, "Teddy Roosevelt was a great President!"

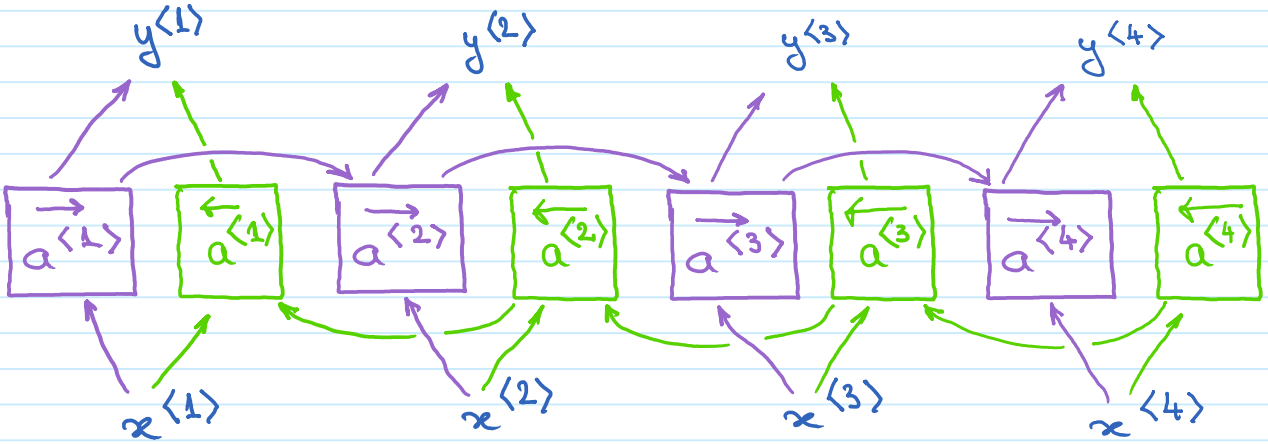


Screen clipping taken: 2018-09-01 17:00

- The networks above is a a unidirectional (or foward directional) RNN used as a name recognition network.
- In this case to determine if $y^{<3>}$ should be flagged as a name requires more information than just the 1st 3 words.
  → this requirement applies regardless of the network being a standard RNN or GRU or LSTM architecture.

BIDIRECTIONAL RNN (BRNN)

Working example has 4 blocks:



→ blocks are part of the forward recurrent component.

← blocks are part of the backward recurrent component.

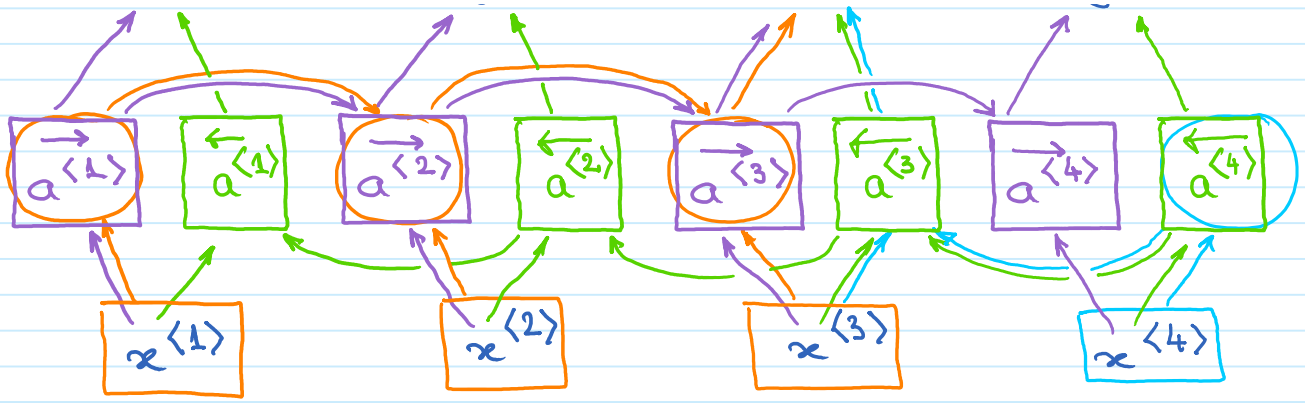
This network defines an acyclic graph.

When computing activations ($\overrightarrow{a}^{\langle t \rangle}$, $\overleftarrow{a}^{\langle t \rangle}$) for the forward and backward recurrent components both passes are part of the forward propagation (i.e. no gradients are computed and no weights are updated)

After both the forward and backward activations have been computed we can output a $\hat{y}$ prediction as follows:

$$\hat{y}^{\langle t \rangle} = g(W_y [\overrightarrow{a}^{\langle t \rangle}, \overleftarrow{a}^{\langle t \rangle}] + b_y)$$

For example to calculate $\hat{y}^{\langle 3 \rangle}$ we use information that flows forward through $\overrightarrow{a}^{\langle 1 \rangle}$, $\overrightarrow{a}^{\langle 2 \rangle}$, $\overrightarrow{a}^{\langle 3 \rangle}$ and information that flows backward through $\overrightarrow{a}^{\langle 4 \rangle}$, $\overrightarrow{a}^{\langle 3 \rangle}$.



- This allows information from the past (upstream of current element in the sequence) and the future (downstream from current element in the sequence) to be used to compute the current element output.

- The activation blocks can be standard RNN, GRU or LSTM units.

- For many Natural Language Processing problems a BRNN with LSTM blocks is commonly used.

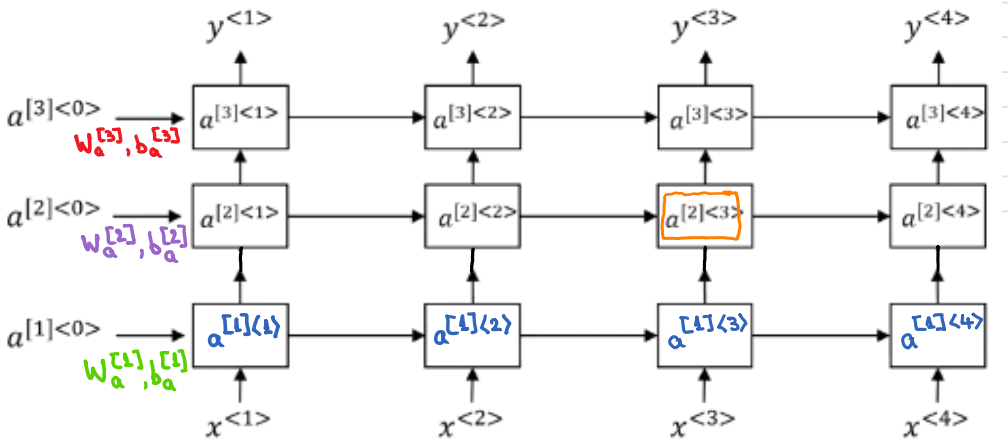
- A disadvantage of BRNNs is that we need to have the entire sequence available before we can compute a predicted output.
  - → so for a speech recognition solution using BRNNs we need to wait for the person to stop talking before we can process the data.
    - → for real time speech recognition networks more complex implementations are used.

Deep RNNs

Saturday, 01 September, 2018 18:58

Goal: Even though the RNNs work well with standard NN for certain problems we need to use Deep RNNs. Here we explore how to implement Deep RNNs.

Standard Deep Network	Deep RNN



Screen clipping taken: 2018-09-01 19:02

$a^{[l]<t>}$ indicates that the input is being proccessed on layer l of the Deep RNN at time t (step t / 2nd dimension layer t)

$a^{[2]<3>}$ has two inputs:

- one from the bottom ($a^{[1]<3>}$)
- one from the left ($a^{[2]<2>}$)

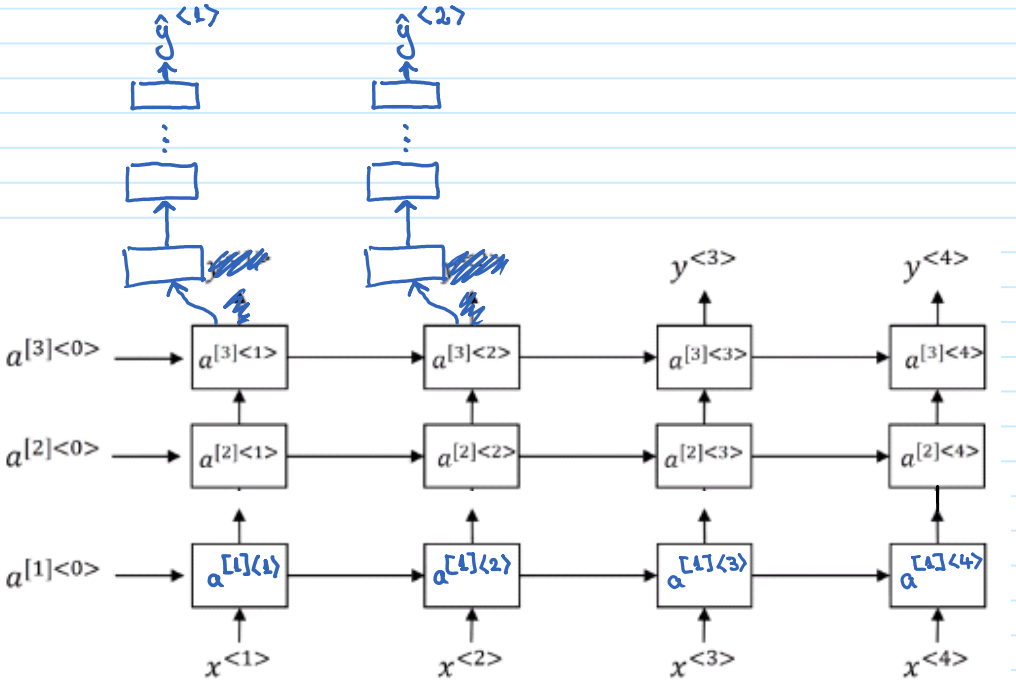
So: $a^{[2]<3>} = g\left(W_a^{[2]}\left[a^{[2]<2>}, a^{[1]<3>}\right] + b_a^{[2]}\right)$

Note: the same $W_a^{[2]}, b_a^{[2]}$ parameters are used for all the steps in layer [2]

- Unlike standard Deep Networks which can have 100s of layers, Deep RNNs

with a depth of 3 are considered quite deep (as we already have another "time" dimension to train through)

- An alternative architecture is to have a "foundation" of Deep RNN units (connected "horizontally" & "vertically"), followed by a standard Deep Network (connected only "vertically") and then the output layer.



- The deep layer of the RNN can use standard RNN units, GRU, LSTM or BRNN architecture.

  → limitation is the compute required to train the network.

Assignment 2 week 1

Sunday, 02 September, 2018 21:03

- **Step 1**: Pass the network the first "dummy" input $x^{\langle 1 \rangle} = \vec{0}$ (the vector of zeros). This is the default input before we've generated any characters. We also set $a^{\langle 0 \rangle} = \vec{0}$
- **Step 2**: Run one step of forward propagation to get $a^{\langle 1 \rangle}$ and $\hat{y}^{\langle 1 \rangle}$. Here are the equations:

$$a^{\langle t+1 \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t \rangle} + b) \tag{1}$$

$$z^{\langle t+1 \rangle} = W_{ya}a^{\langle t+1 \rangle} + b_y \tag{2}$$

$$\hat{y}^{\langle t+1 \rangle} = softmax(z^{\langle t+1 \rangle}) \tag{3}$$

Note that $\hat{y}^{\langle t+1 \rangle}$ is a (softmax) probability vector (its entries are between 0 and 1 and sum to 1). $\hat{y}_i^{\langle t+1 \rangle}$ represents the probability that the character indexed by "i" is the next character. We have provided a `softmax()` function that you can use.

- **Step 3**: Carry out sampling: Pick the next character's index according to the probability distribution specified by $\hat{y}^{\langle t+1 \rangle}$. This means that if $\hat{y}_i^{\langle t+1 \rangle} = 0.16$, you will pick the index "i" with 16% probability. To implement it, you can use `np.random.choice`.

Here is an example of how to use `np.random.choice()`:

```
np.random.seed(0)
p = np.array([0.1, 0.0, 0.7, 0.2])
index = np.random.choice([0, 1, 2, 3], p = p.ravel())
```

This means that you will pick the `index` according to the distribution: $P(index = 0) = 0.1, P(index = 1) = 0.0, P(index = 2) = 0.7, P(index = 3) = 0.2$.

- **Step 4**: The last step to implement in `sample()` is to overwrite the variable x, which currently stores $x^{\langle t \rangle}$, with the value of $x^{\langle t+1 \rangle}$. You will represent $x^{\langle t+1 \rangle}$ by creating a one-hot vector corresponding to the character you've chosen as your prediction. You will then forward propagate $x^{\langle t+1 \rangle}$ in Step 1 and keep repeating the process until you get a "\n" character, indicating you've reached the end of the dinosaur name.

Screen clipping taken: 2018-09-02 21:04

2.2 - Sampling

Now assume that your model is trained. You would like to generate new text (characters). The process of generation is explained in the picture below:

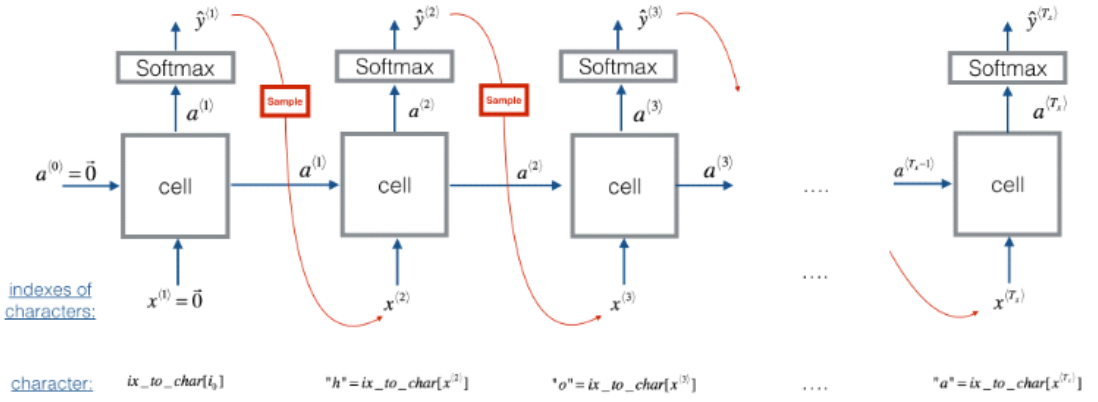


Figure 3: In this picture, we assume the model is already trained. We pass in $x^{\langle 1 \rangle} = \vec{0}$ at the first time step, and have the network then sample one character at a time.

Exercise: Implement the `sample` function below to sample characters. You need to carry out 4 steps:

Screen clipping taken: 2018-09-02 21:04

Assignment 3 Week 1

Sunday, 02 September, 2018 23:37

Exercise: Implement `djmodel()`. You will need to carry out 2 steps:

1. Create an empty list "outputs" to save the outputs of the LSTM Cell at every time step.
2. Loop for $t \in 1, \ldots, T_x$:

   A. Select the "t"th time-step vector from X. The shape of this selection should be (78,). To do so, create a custom Lambda layer in Keras by using this line of code:

   ```
   x = Lambda(lambda x: X[:,t,:])(X)
   ```

   Look over the Keras documentation to figure out what this does. It is creating a "temporary" or "unnamed" function (that's what Lambda functions are) that extracts out the appropriate one-hot vector, and making this function a Keras `Layer` object to apply to `X`.

   B. Reshape x to be (1,78). You may find the `reshapor()` layer (defined below) helpful.

   C. Run x through one step of LSTM_cell. Remember to initialize the LSTM_cell with the previous step's hidden state a and cell state c. Use the following formatting:

   ```
   a, _, c = LSTM_cell(input_x, initial_state=[previous hidden state, previous cell state])
   ```

   D. Propagate the LSTM's output activation value through a dense+softmax layer using `densor`.

   E. Append the predicted value to the list of "outputs"

Screen clipping taken: 2018-09-02 23:37

Exercise: Implement the function below to sample a sequence of musical values. Here are some of the key steps you'll need to implement inside the for-loop that generates the T_y output characters:

Step 2.A: Use `LSTM_Cell`, which inputs the previous step's `c` and `a` to generate the current step's `c` and `a`.

Step 2.B: Use `densor` (defined previously) to compute a softmax on `a` to get the output for the current step.

Step 2.C: Save the output you have just generated by appending it to `outputs`.

Step 2.D: Sample x to the be "out"'s one-hot version (the prediction) so that you can pass it to the next LSTM's step. We have already provided this line of code, which uses a Lambda function.

```
x = Lambda(one_hot)(out)
```

[Minor technical note: Rather than sampling a value at random according to the probabilities in `out`, this line of code actually chooses the single most likely note at each step using an argmax.]

Screen clipping taken: 2018-09-03 00:01

Week 2 NLP & Word Embeddings

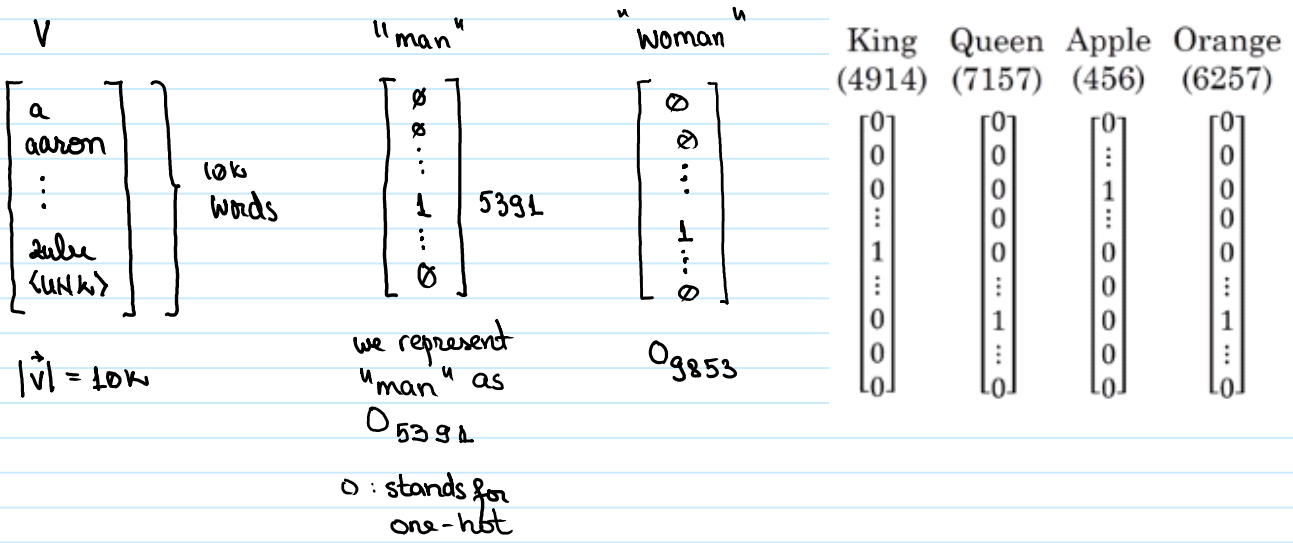
Monday, 03 September, 2018 00:46

Word representation

Monday, 03 September, 2018 19:46

Goal: Learn about what the limitations of one-hot encoding representations are and how to use word embeddings to allow NN to understand analogies in bodies of text (i.e "man is to woman as king is to queen")

So far we have been representing words using one-hot encoded vector vocabularies.



ONE-HOT WEAKNESS:

One-hot representations treat each word as an independent entity. This doesn't allow algorithms to establish relationships between words. It is not easy for algorithms to learn to generalize accross words.

- "apple & orange" are more related than "apple & king"
- If the algorithm learns to infer one pairing it doesn't know how to generalize

I want a glass of orange juice

I want a glass of apple ? (can't generalize)

This limitation is caused by the fact that the inner product (dot product) between any two words represented as one-hot encoding is 0. Also the distance between any two words is the same for all pairs in the vocabulary.

(i.e. there is no way to represent the relationship between apple & orange is closer than apple & king).

FEATURIZED REPRESENTATION (AKA WORD EMBEDDING)

- An alternative to one-hot representation is featurized representation.
- In this representation we attach a score to each word that indicates how much they match a feature

these two words have similar feature vectors.

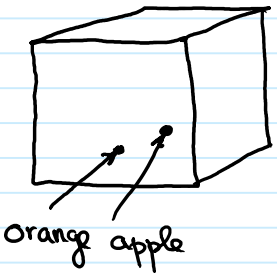
Feature	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
⋮	⋮	⋮				
size						
cost						
alive						
noun	↓	↓				
	e_{5391}	e_{9853}				

Screen clipping taken: 2018-09-03 21:08

- Let's say we come up with 300 features to rate each word with
- So each word in our vocabulary has now associated with it a 300 dimensional vector. We will use $e_{<\text{vocab. index}>}$ to identify these feature vectors.

  i.e. man's feature vector is e_{5391}.
- e stands for embeddings. We call the 300D vectors embeddings because

each word is embedded somewhere in the 300 dimensional space.



if it was a 3D instead of 300D space

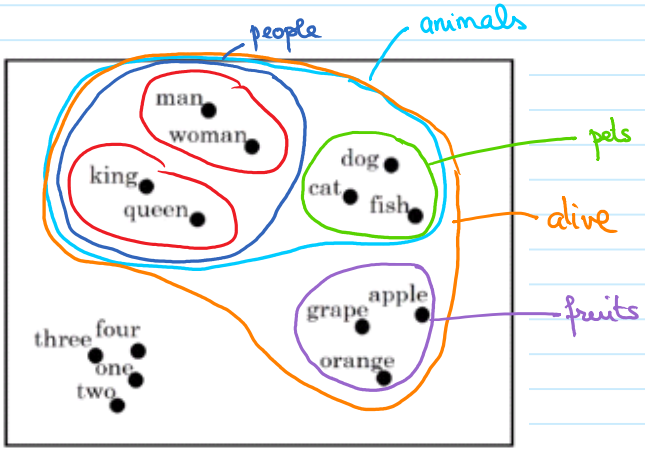
- As we train the NN the features that will emerge will not be as easily interpretable as the features above but they will have the same function (allows to establish how much words are related)
- We will use both one-hot and feature representations in our products.

VISUALIZING WORD EMBEDDINGS:

Take a 300 Dimensional embedding and visualize it in 2 dimensions using the t-SNE algorithm.

[van der Maaten and Hinton., 2008. Visualizing data using t-SNE]

Screen clipping taken: 2018-09-03 21:29



Screen clipping taken: 2018-09-03 21:27

- This indicates that the algorithm can learn there exists relationships between words.

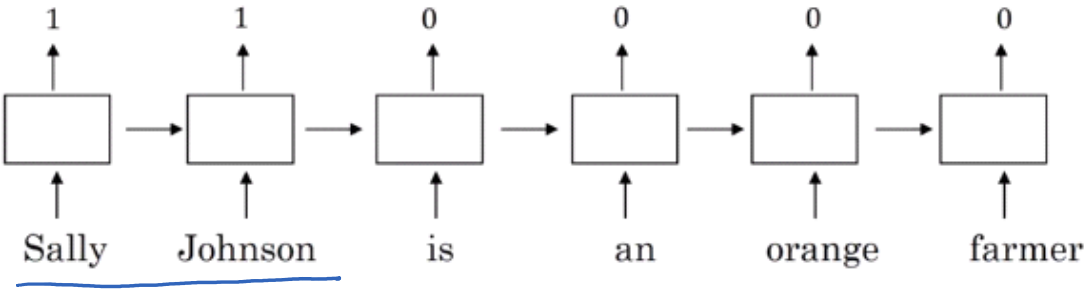
Using word embeddings

Monday, 03 September, 2018 23:00

Goal: Learn how to apply word embeddings to Natural Language Processing problems.

NAMED ENTITY RECOGNITION WORKING EXAMPLE

Given a sentence identify which words are part of names.



Screen clipping taken: 2018-09-03 22:11

* Note for simplicity this is a unidirectional NN. In practice we would use a bidirectional NN.

In this case if the NN determines that the pair of elements "Sally Johnson" is highly likely to be a person when it is associated with "orange farmer" it can then perform a similar classification for the sentence below and identify "Robert" & "Lin" as being name words

Screen clipping taken: 2018-09-03 23:02

knowing that "apple" & "orange" have similar word embedding vectors.

- What if the NN instead encounters the more exotic sentence below:

Robert Lin is a durian cultivator.

* durian is south-east asian fruit

For a small training set the words "durian" and "cultivator" [illegible]

For a small training set the words "durian" and "cultivator" might not even be present.

But if the NN determines that "durian" is kind of like a fruit and "cultivator" is kind of like a farmer then it might still be able to generalize that "Robert" and "Lin" are part of names because "durian cultivator" is most likely a person.

asian fruit

Screen clipping taken: 2018-09-03 23:12

This is possible to do using word embeddings because the algorithms for training word embeddings are oftened trained on data sets of billions of words (maybe even 100B words) - available almost for free on the internet.

→ So even if the vocabulary that is used for the Name Entity Recognition Task NN is small (say 10k words like in Notation slides) the word embeddings associated with these 10k words are very well developed.

→ This is a type of transfer learning where we use the information developed during the training of a NN using vast training data sets to augment a NN that is much less powerful (but maybe faster).

TRANSFER LEARNING & WORD EMBEDDINGS:

1. Learn word embeddings from large text corpus. (1-100B words)

   (Or download pre-trained embedding online.)

2. Transfer embedding to new task with smaller training set. (say, 100k words)

   - we can maybe use much lower dimensional feature vector
     i.e instead of using a sparse 10k dimensional one-hot encoded vector we can use a 300 dimensional dense vector (that we learned through word embeddings)

3. Optional: Continue to finetune the word embeddings with new data.

Screen clipping taken: 2018-09-03 23:39

→ we would only do the fine tunning if task 2 has a large data set

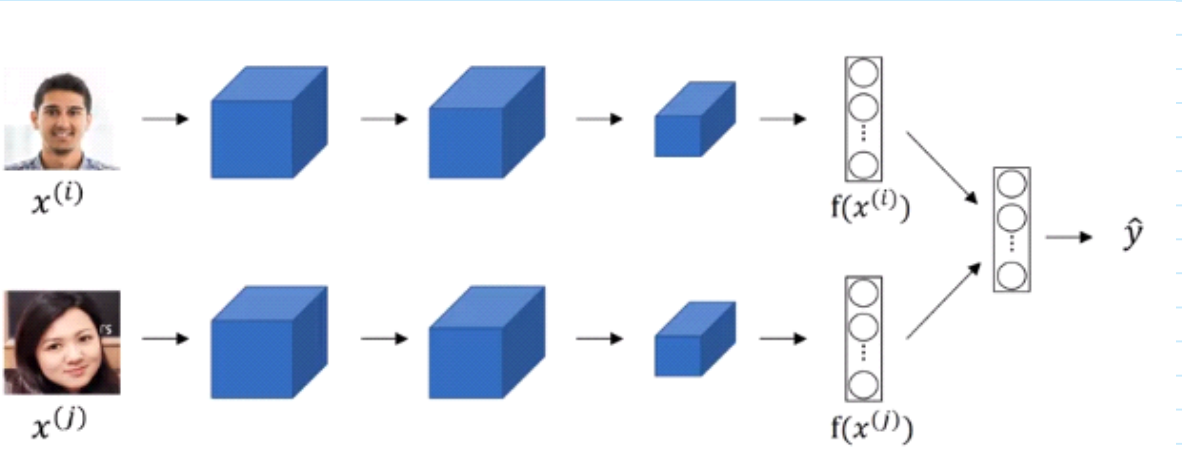
- Word embeddings are most effective when the training task has a small training set.
  i.e. If transfer learning occurs from task A to B it is most effective when task A has a large training set & task B has a small training set.

So word embeddings are

useful for: name entity recognition, text summarization, code reference resolution, parsing

not so useful for: language modelling, machine translation (where the training data sets are very large)

WORD EMBEDDING RELATION TO FACE ENCODING:



[Taigman et. al., 2014. DeepFace: Closing the gap to human level performance] Andrew Ng

Screen clipping taken: 2018-09-04 00:53

For face recognition we trained a Siamese Network architecture that learns a 128D encoding for each face and uses these encodings to compare faces.

The encoding is similar to an embedding

The word embeddings are different in that the vocabulary is fixed (say 10k words). We can only learn an encoding for these 10k words ($e_1, \ldots e_{10,000}$)

For face recognition we can generate an encoding even for a face we have never seen before.

— LIMITED VOCABULARY VS UNLIMITED FACES —

Properties of word embeddings

Tuesday, 04 September, 2018 15:00

Goal: Learn about properties of WE that help with reasoning about analogies.

ANALOGIES

[Mikolov et. al., 2013, Linguistic regularities in continuous space word representations]

Screen clipping taken: 2018-09-04 15:05

Features	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	−1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97
	e_{man} e_{5391}	e_{woman}				

Screen clipping taken: 2018-09-04 15:03

Let's say we pose the question:

Q: Man is to woman as king is to what?

A: Queen.

Is it possible for an algorithm to figure this out?

For this problem assume we are using only 4D word embeddings (like in table above).

We name the man and woman word embeddings as e_{man} and e_{woman} respectively

$$e_{man} - e_{woman} = \begin{bmatrix} -1 \\ 0.01 \\ 0.03 \\ 0.09 \end{bmatrix} - \begin{bmatrix} 1 \\ 0.02 \\ 0.02 \\ 0.01 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$e_{king} - e_{queen} = \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

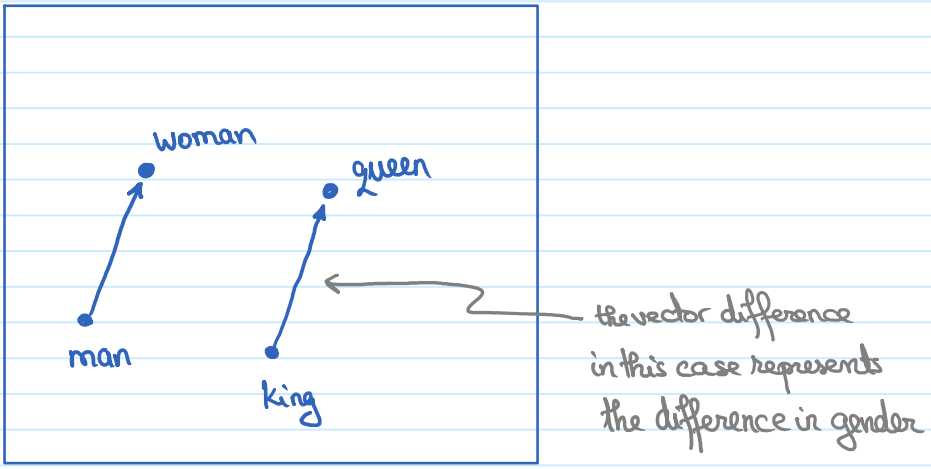
These two differences of word embeddings

$$e_{man} - e_{woman} \approx e_{king} - e_{queen}$$

indicate that in both cases the result is mainly gender.

- So to resolve the analogy the algorithm can iterate through all the words and compute the difference from e_{king} for each one of them trying to find a result that matches the difference $e_{man} - e_{woman}$

FORMAL ANALOGY ALGORITHM DEFINITION:



t-SNE 2D plot of 300D word embeddings

We want to find the word w so that the equation below is true:

$$e_{man} - e_{woman} \approx e_{king} - e_w$$

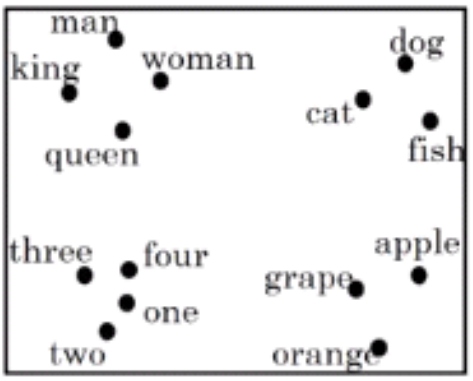
More specifically find w that maximizes the similarity between e_w and

$(e_{king} - e_{man} + e_{woman})$:

$$\arg\max_w \; sim(e_w, e_{king} - e_{man} + e_{woman})$$

The remarkable thing is that this optimization works (analogy classifiers have a 30-75% accuracy - an analogy is correct if the exact word is predicted).

NOTES ON T-SNE PLOTS:



Screen clipping taken: 2018-09-04 15:32

Take a 300 dimensional data set and map it (in a very non-linear way) on 2D space.

The vector differences we saw in the formal definition above will not look as elegant as depicted as the t-SNE algorithm will severely distort the mapping. The "parallel" vectors only exist in 300D.

COSINE SIMILARITY FUNCTION

$$sim(e_w, e_{king} - e_{man} + e_{woman})$$

Screen clipping taken: 2018-09-04 15:37

Define similarity between two vectors u & v as:

$$Sim(u, v) = \frac{u^T v}{\|u\|_2 \, \|v\|_2}$$

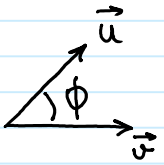
$u^T v$ ← inner product (dot product)

$\|u\|_2$ ← Euclidean length

If u & v are similar their inner product is large.

$\vec{u}$

Called "cosine similarity" because $\cos\phi = \frac{u^T v}{\|u\|_2 \|v\|_2}$ for:



Remember:

- We could use "disimilarity" instead $\|u-v\|^2$ but cosine similarity is used more frequently (as the cosine similarity normalizes better the lenghts of vectors u & v).

- One remarkable property of WE is how generalizable the learning is:

Man:Woman as Boy:Girl
Ottawa:Canada as Nairobi:Kenya
Big:Bigger as Tall:Taller
Yen:Japan as Ruble:Russia

Screen clipping taken: 2018-09-04 15:46

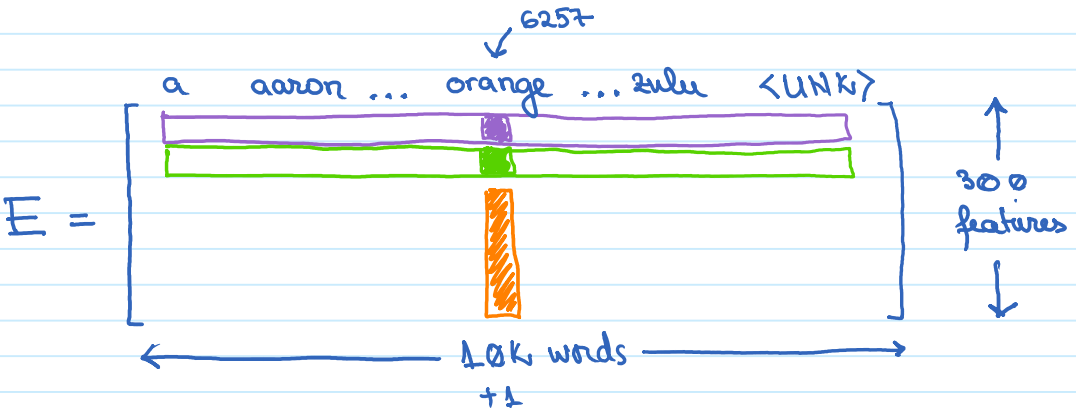
These analogies are learned automatically by training the WE classifier on large bodies of text.

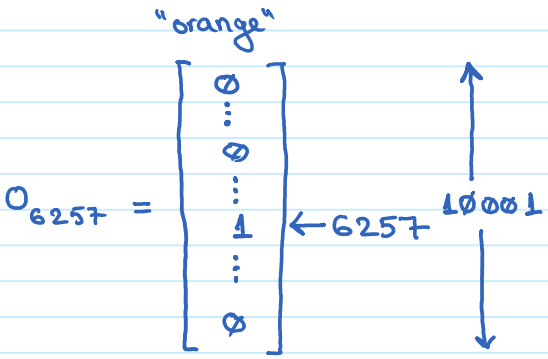
Embedding matrix

Tuesday, 04 September, 2018 16:02

Goal: Learn the concept of an Embedding Matrix used in learning Word Embeddings.

- For this section we will use the 10K words vocabulary.
- The algorithm we design will learn a 300 x 10K dimensional matrix of word embeddings we will call E as below





Taking the inner product of E with O_{6257} is similar to masking E such that column 6257 is outputted (this is because all other values are zeroed out as O_{6257} is 0 with the exception of row 6257 which is 1)

$$\underset{(300,10001)}{E} \cdot \underset{(10001,1)}{O_{6257}} = \begin{bmatrix} \\ \\ \\ \end{bmatrix} = e_{6257}$$

(300, 1)

In general:

e_w = embedding for word w

$E \cdot o_j = e_j$

The approach is to initialize E randomly & then use Gradient Descent to learn the parameters of the 300 × 10k matrix.

NOTE: In practice it is not efficient to use the inner product to pull out the e_j embedding from matrix E (we multiply too many elements by 0). We will use specialized functions to look up an embedding.

i.e. Kavius: Use the Embedding layer.

Learning word embeddings

Tuesday, 04 September, 2018 16:43

Goal: Learn concrete algorithms that learn word embeddings.

Historical note: Initially the algorithms for learning Word Embeddings were quite complex, but as researchers iterated they simplified the algorithms to the point that the most recent algorithms feel "magical" in how they can learn.

To get a better intuition for how the recent algorithms work we will first review the older more complex algorithms

HISTORICAL NEURAL LANGUAGE MODEL

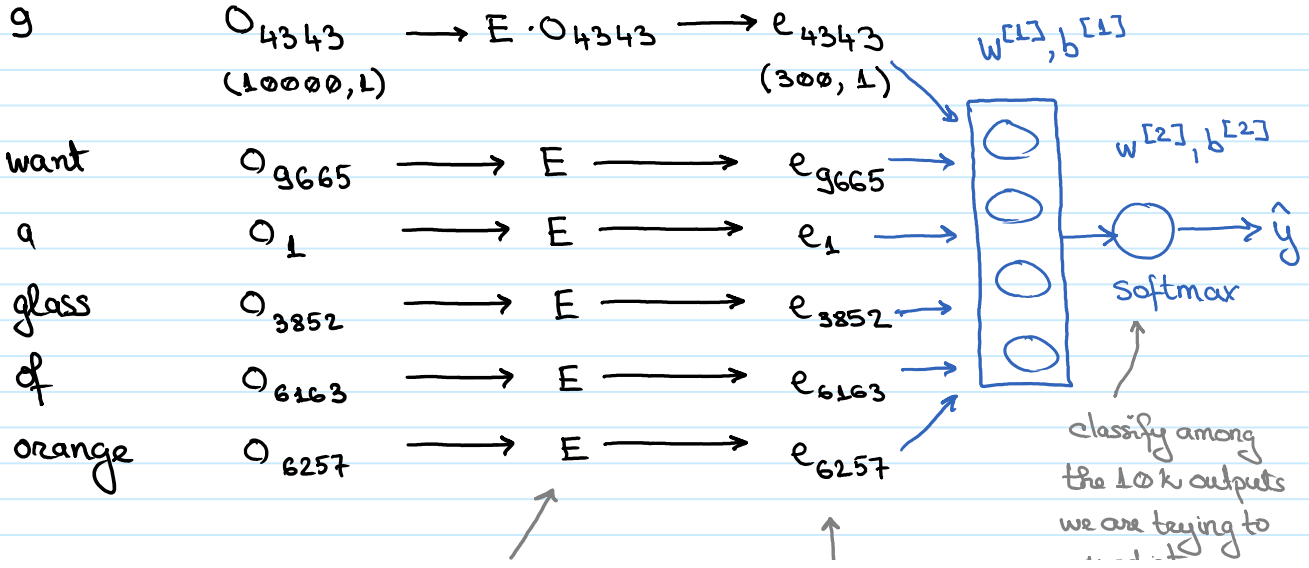
[Bengio et. al., 2003, A neural probabilistic language model]

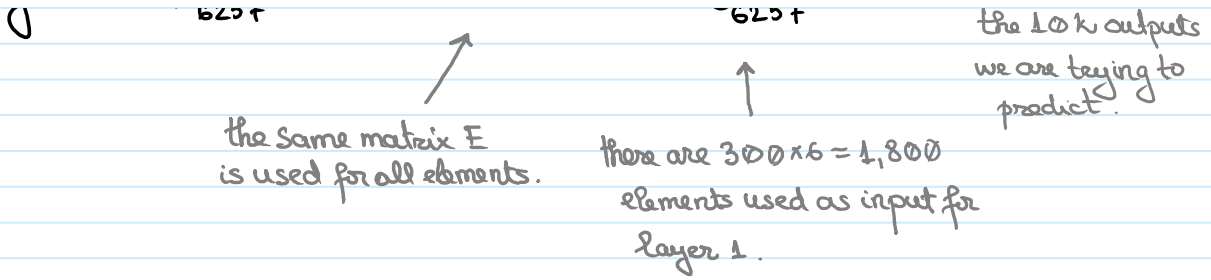
Screen clipping taken: 2018-09-04 16:58

Given a sequence of words predict the next word in the sequence.

I want a glass of orange ____.

O: 4343 9665 1 3852 6163 6257

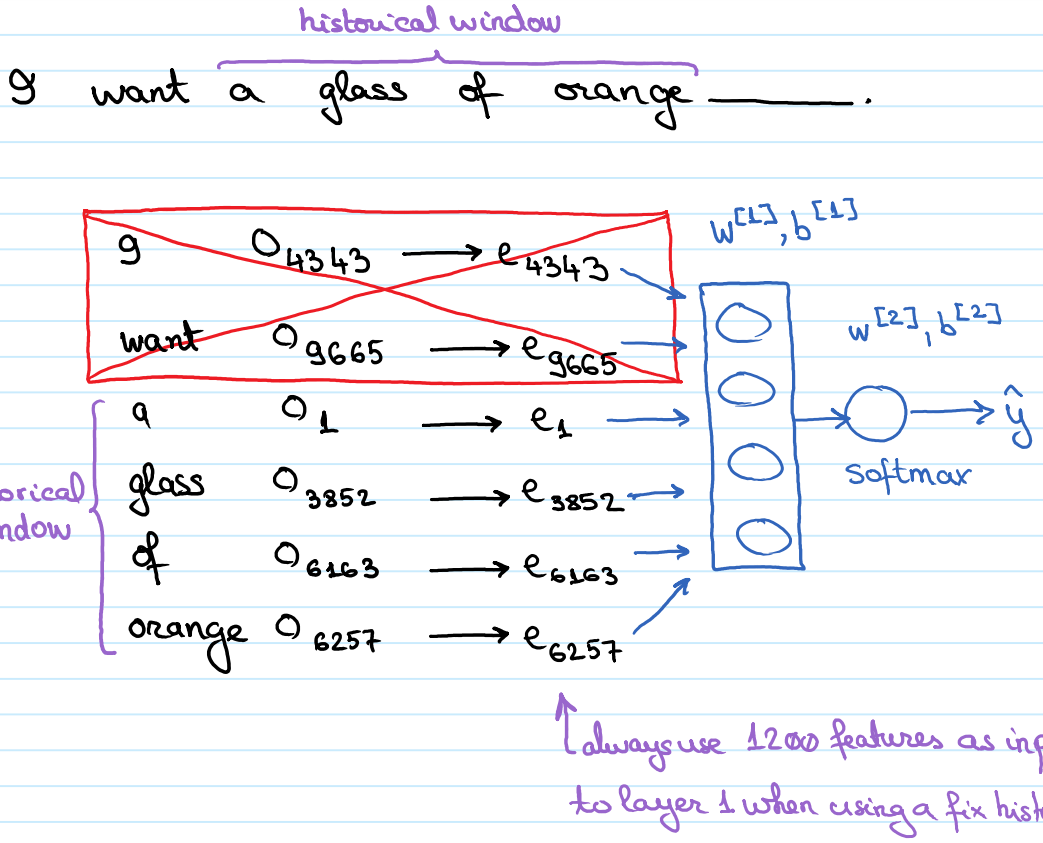




A common technique to adjust the number of elements used as input into layer 1 is to use a fixed historical window

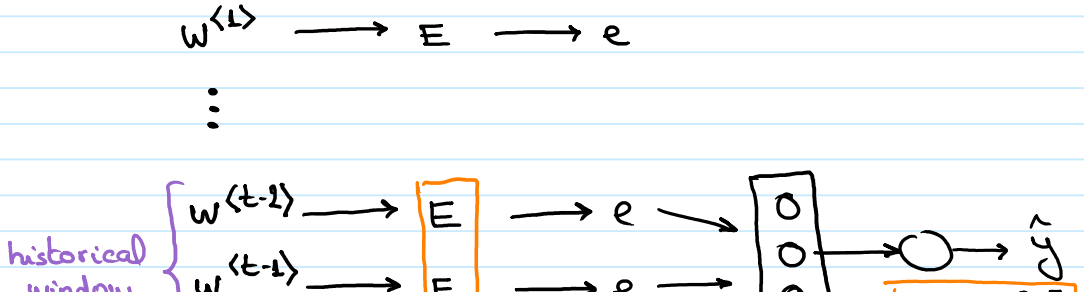
→ i.e. look back at the previous 4 words to predict the current word.

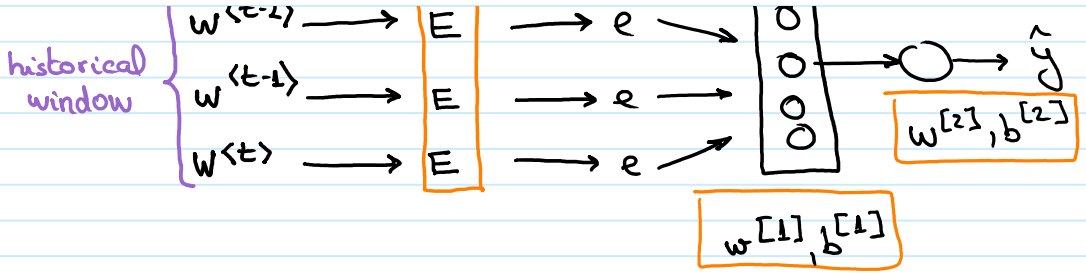
↑ hyperparameter.



↑ always use 1200 features as input to layer 1 when using a fix history.

So the parameters for the model are: matrix E, $W^{[1]}, b^{[1]}$, $W^{[2]}, b^{[2]}$





We will use Gradient Descent to optimize these parameters so that we maximize the likelyhood of predicting the next word in the training set given a sequence preceeding it.

In this case the Gradient Descent will push the classifier to learn similar embeddings for "apple" and "orange" as it will "connect" the two elements when it often tries to predict "apple juice" or "orange juice".

→ the same for "apple", "oranges", "grapefruit", "pears" etc.

OTHER CONTEXT / TARGET PAIRS:

For the next section we want to use a more complex sentence as an example:

I want a glass of orange juice to go along with my cereal.

context

target (what is to be predicted)

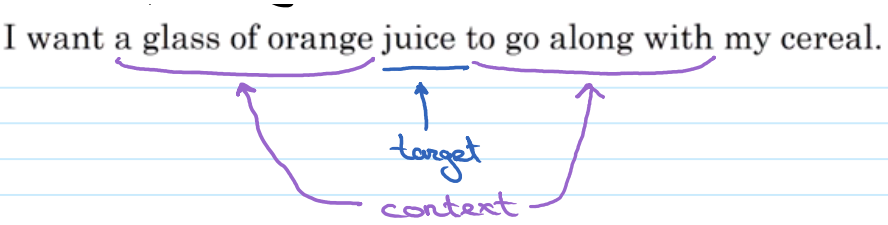
Screen clipping taken: 2018-09-04 17:44

In this case the context is the last 4 words before target

If the goal is to build a language model it is common to use as context a few words before the target. But for other goals we can use other contexts.

- Last 4 words (see above)
- 4 words on left & right.

I want a glass of orange juice to go along with my cereal



in this case the input into the word embedding learning algorithm is:

a glass of orange

to go along with

- Last 1 word (a simpler context):

  input: orange

- Nearby 1 word (aka SKIP GRAM)

  We encountered the word "glass" sometimes earlier what is a likely word to be used in this context.

So if we want to learn a Word Embedding our context can be more simpler than when we want to learn a language model.

Word2Vec

Tuesday, 04 September, 2018 20:08

Goal : Learn about Word 2Vec - a more recent, simpler & computationally more efficient algorithm for learning Word Embeddings.

[Mikolov et. al., 2013. Efficient estimation of word representations in vector space.]

Screen clipping taken: 2018-09-04 20:31

SKIP-GRAMS

Let's start with the following sentence from our training set:

I want a glass of orange juice to go along with my cereal.

target context target target

Screen clipping taken: 2018-09-04 20:33

If we use the Skip Gram model we need to come up with a few (context, target) pairs to define our supervised learning problem:

→ this is different than using the last n-words before the target as context

We can randomly pick a few words to be our context words and then choose another word within "n-words away" (say +/- 5 words) of the context word to be the target word.

Context	Target
orange	juice
orange	glass
orange	my

The goal is to come up with a good training data set so we can learn good Word Embeddings.

THE SKIP-GRAM MODEL:

- Skip-gram refers to the fact that we are randomly skipping words on which we are training
- For this problem we will use a Vocab of 10k words (some train on Vocabs that are 1M+ words).
- The problem is to learn a mapping from context element to target element.

Context c ("orange") → Target t ("juice")

O_{6257} → O_{4834}

X → Y

We will use the following skip-gram architecture

$$O_c \rightarrow E \rightarrow e_c \rightarrow \boxed{\text{SOFTMAX}} \rightarrow \hat{y}$$

where

O_c = one-hot encoding of context word

E = Embedding matrix (parameters to be learned)

$e_c = E \cdot O_c$ (word embedding associated with context word c i.e. a column in matrix E)

Where SOFTMAX:

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10k} e^{\theta_j^T e_c}}$$

probability of getting

probability of getting target word t given input word c

$j=1$

θ_t : parameter associated with output t (what is the chance of a particular word t being the label). θ_t are parameters of the SOFTMAX layer.

We define the loss function as:

$$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{10000} y_i \log \hat{y}_i$$

where ground truth target y & predicted output target $\hat{y}$ are represented as a one-hot vector

"juice"

$$y = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow 4834$$

PROBLEMS WITH SOFTMAX CLASSIFICATION

- Problem 1: Computational speed

  Every time we calculate a probability we need to compute a sum over all the words in the vocabulary (10K to 1M+ words depending on the vocabulary).

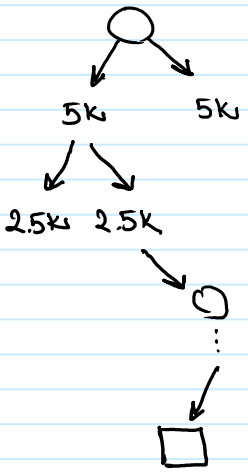
$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

Screen clipping taken: 2018-09-04 21:09

This makes scaling to larger vocabularies a very slow proposition.

Solution : Use a Hierarchical SOFTMAX

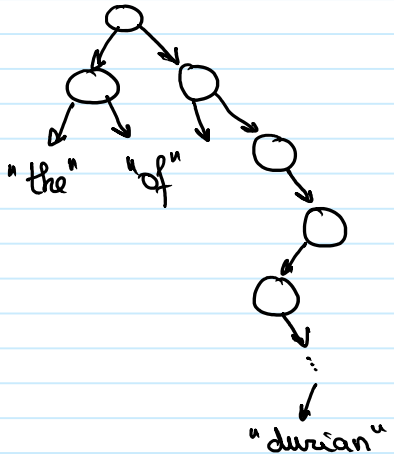
→ instead of trying to classify the output into all the classes available, have SOFTMAX indicate if the output is in the 1st 5000 or the 2nd 5000 classes.



Using a symmetric binary tree scales the computational cost by $O(\log n)$ instead of $O(n)$ when summing

→ where n is the Vocab size

In practice though the tree is not designed to be perfectly balanced (perfectly symmetric). Instead the more common words are at the top of the tree while less common words (i.e. "durian") are deeper.



More details on this in the paper.

[Mikolov et. al., 2013. Efficient estimation of word representations in vector space.]

Another solution is the "Negative Sampling".

- Problem 2: How to sample the context c?

  If we sample c randomly most of the words we will select are the

common words like : "the", "of", "a", "and", "to" so we would spend most of our time updating the embeddings e_c associated with these words

We ideally also sample the less frequent words like "orange", "apple", "durian" so we update their embeddings.

Solution : We use heuristics to bias the random sampling so we sample both common and less common words.

Note: The Mikolov paper also describes another model: CBOW (continuous backwards model).

Summary: The SOFTMAX is computationally expensive.

Negative Sampling

Tuesday, 04 September, 2018 23:33

Goal: Learn how to address the computational inefficiency of the SOFTMAX step in the Skip Gram models for training Word Embeddings using Negative Sampling

[Mikolov et. al., 2013. Distributed representation of words and phrases and their compositionality]

Screen clipping taken: 2018-09-04 23:39

To understand how Negative Sample works we need to define a new learning problem.

NEW PROBLEM DEFINITION

Given a pair of context-word we are going to predict if this is a context-target pair.

X		Y
Context	Word	target?
"orange"	"juice"	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0 ←

K times sample randomly from vocab to pick negative examples here $K = 4$

"of" actually appears in close proximity to "orange" in the training input sentence (i.e. it is part of the context) It is OK to mislabel it as there won't be many of them

The positive entries are obtained by the standard Skip-gram method where we sample the input sentence randomly for a word (which we label as context) and then we sample in close proximity (use a window of +/- 5 words) to the target for another word (which we label as target).

For the negative entries we perform the same steps except for the target word we choose a random word from the Vocabulary (under the assumption that they are not related with an actual target).

→ repeat negative examples steps so we have k negative examples.

$k = 5\text{-}20$ for small data sets

$k = 2\text{-}5$ for large data sets

Use the (X, Y) training set data we generated above to train a neural net so that it predicts if the two words were chosen at random from the dictionary or if they were in close proximity to each other in the training set sentences (it tries to tell which distribution we sampled from).

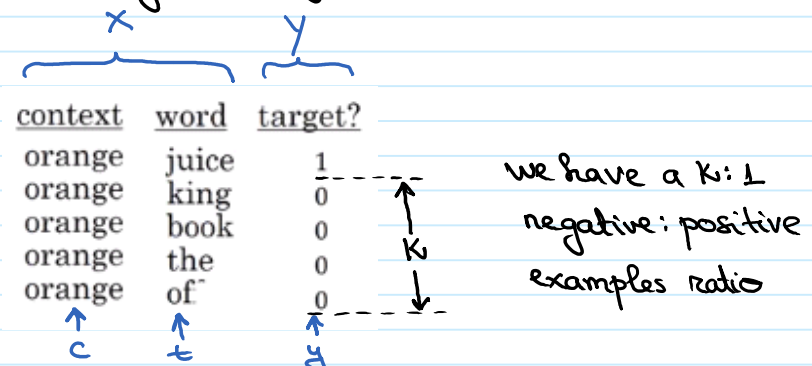
NEGATIVE SAMPLE MODEL:

Previous SOFTMAX model we used was:

$$\text{Softmax:} \quad p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

Screen clipping taken: 2018-09-05 00:22

The training set we generated was:



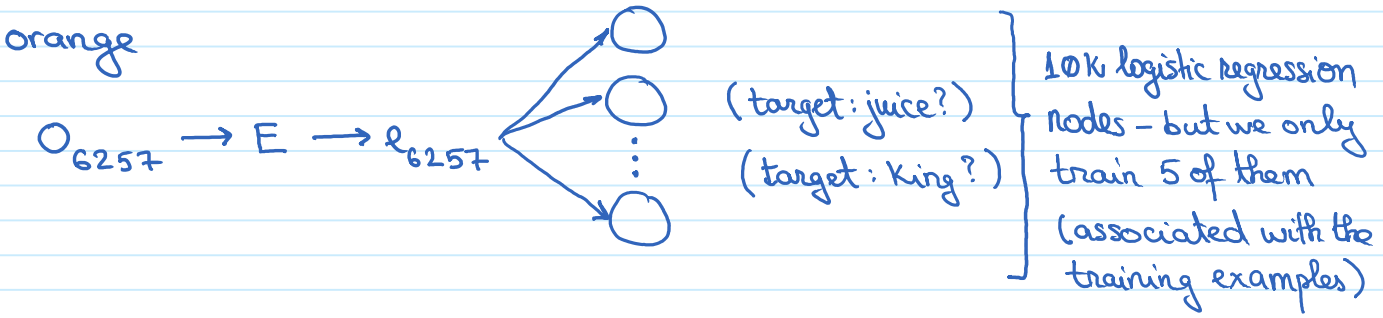
Screen clipping taken: 2018-09-05 00:23

We'll model this as a logistic regression:

$$P(y==1 \mid c,t) = \sigma(\theta_t^T e_c)$$

θ_t: parameter for each target word

e_c: parameter (the embedding vector) for each context word.



So instead of having a computationally expensive 10,000 classes SOFTMAX layer we use a layer with 10,000 binary classification nodes (each of which is trained easily) and in each iteration we are only going to train 5 of these nodes (4 negative examples + 1 positive example)

→ training $k+1$ nodes is cheaper than training a 10k classes SOFTMAX

This technique is called Negative Sampling as for each positive example we choose k negative examples.

SELECTING NEGATIVE EXAMPLES

context (x)	word (x)	target? (y)
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	

How do we sample the negative examples?

- Sample according to the empirical frequency of words in your corpus.
  - Problem: we have a high incidence of "the",

orange	king	0
orange	book	0
orange	the	0
orange	of	0
↑ c	↑ t	↑ y

words in your corpus.

- Problem: we have a high incidence of "the", "of", "and"

- Sample using a $\frac{1}{|V|}$ uniformly at random ($|V|$ ≈ vocab size)

- Problem: this is not representative of the words in English.

- Sample using an in between the two heuristic:

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10000} f(w_j)^{3/4}}$$

$f(w_i)$ ← observed frequency of word w_i in the English language.

$P(w_i)$ ↑ probability a word is sampled

By taking the power of 3/4 causes the sampling distribution to fall in between uniform randomly distributed & empirical English language distribution.

Empirically this appears to work well according to Mikolov et al.

Note: There are pre-trained Word Embedding vectors online that can be used readily.

GloVe word vectors

Wednesday, 05 September, 2018 14:43

Goal: Learn about GloVe (Global Vectors for Word Representation) algorithm - another way of computing Word Embeddings.

Note: This algorithm is not used as much as Word2Vec or Skip Gram algorithms but it is sometimes used because of its simplicity.

For more details see:

[Pennington et. al., 2014. GloVe: Global vectors for word representation]

Screen clipping taken: 2018-09-05 14:48

GLOVE WORKING EXAMPLE

I want a glass of orange juice to go along with my cereal.

Screen clipping taken: 2018-09-05 14:51

For the previous Word2Vec algorithm we were sampling word pairs (context, target) by picking words that appeared in close proximity to each other.

GloVe algorithm makes the counting of these sampled pairs explicit. It iterates through the training set corpus and counts how many times the word i (target) appears in the proximity of word j (context).

$X_{i,j}$ = number of times i appears in the context of j

($X_{t,c}$ = number of times t appears in the context of c)

Depending on how we define "context" $X_{i,j} = X_{j,i}$

i.e. • if context is a window of +/- k words around the target
then $X_{i,j} = X_{j,i}$

• if context is the word immediately before the target $X_{i,j} \neq X_{j,i}$

GLOVE MODEL

Weighing term accounting for:
- degenerate pairs ($X_{i,j} = 0$)
- frequent pairs
- infrequent pairs

$$\text{Minimize} \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{i,j}) \left(\theta_i^T e_j + b_i + b_j' - \log X_{i,j} \right)^2$$

($\theta_i^T e_j$: t = i, c = j; b_i: t = i; b_j': c = j)

homologous to "$\theta_t^T e_c$"

an indication of how related words i & j are as measured by how often they occur close together as indicated by $X_{i,j}$

- We will use Gradient Descent to find the parameters of θ & e which minimize the sum
- Notice that θ_i & e_j are symmetric ($\theta_i^T e_j = e_j^T \theta_i$). They play a very similar role in achieving the optimization objective.

So one way to train the network is:

1) initialize θ & e uniform randomly
2) run Gradient Descent

2) run Gradient Descent

3) for every word set $e_w^{(final)} = \frac{e_w + \theta_w}{2}$

In previous algorithms θ & e were not playing the same role and we couldn't average them like this

purpose of $f(X_{i,j})$

1) For pairs of words where we don't find the target anywhere in the context $X_{i,j} = 0$. This causes $\log X_{i,j} = -\infty$ (undefined)

By convention we set $f(x_{i,j}) = 0$ when $x_{i,j} = 0$ and we also by convention set $0 \log 0 = 0$

This means we ignore word pairs that are "not related" (i.e. the target is not in context) when we calculate the sum

2) Another usage for $f(X_{i,j})$ is to assign different weights to different "groups" of words:

- assign a lower weight (decrease the computational importance) of stop words — words that appear very often in the English language

  i.e. this, is, of, a

- assign a higher weight (importance) to infrequent words

  i.e. durian

There are various heuristic on choosing $f(X_{i,j})$ in the paper.

This GloVe algorithms appears to simple to work

How can it be that just minimizing a square cost function allows us to learn meaningful word embeddings?

But it works and is the distillation of many iterations of research and actually helped simplify our understanding of earlier (more complex) algorithms.

NOTE ON FEATURIZATION VIEW OF WORD EMBEDDINGS

Features ↓	Man (5391)	Woman (9853)	King (4914)	Queen (7157)
Gender	−1	1	-0.95	0.97
Royal	0.01	0.02	0.93	0.95
Age	0.03	0.02	0.70	0.69
Food	0.09	0.01	0.02	0.01

Screen clipping taken: 2018-09-06 00:04

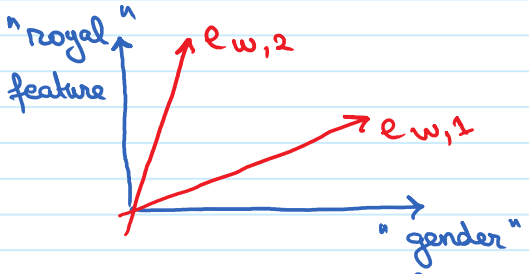
When we learn a Word Embedding using algorithms we talked about i.e. GloVe:

$$\text{minimize} \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij})\left(\theta_i^T e_j + b_i - b_j' - \log X_{ij}\right)^2$$

Screen clipping taken: 2018-09-06 00:06

we cannot guarantee that the individual components of the embeddings are interpretable (the axis upon which we represent the features are not guearanteed to be human interpretable - specifically the first axis might be a combination of "gender", "royal", "age" & "food").

Say we have a space as follows:



feature

we cannot guarantee that the words 1st axis of the Embedding Vector is aligned with the "royal" axis.

→ learning algorithm might choose different axis for these dimensions (axis that are not necessarily orthogonal)

Linear Algebra intuition:

We have a potentially arbitary linear transformation of features.

→ if there is an invertable matrix A then we could replace

$\theta_i^T e_j$ with $(A\theta_i)^T (A^{-T} e_j) = \theta_i^T A^T A^{-T} e_j = \theta_i^T e_j$

! Remember though the "parallelogram math" we used to describe analogies still works even for these non-human interpretable axis.

Sentiment Classification

Sunday, 09 September, 2018 10:14

Goal: Learn how to perform sentiment classification - process a body of text and determine if the topic of discussion is "liked" or "disliked".

SENTIMENT CLASSIFICATION PROBLEM:

CHALLANGE: Small training set.

SOLUTION: Use Word Embedding classifiers to address problem

X	Y
The dessert is excellent.	★★★★☆
Service was quite slow.	★★☆☆☆
Good for a quick meal, but nothing special.	★★★☆☆
Completely lacking in good taste, good service, and good ambience.	★☆☆☆☆

Screen clipping taken: 2018-09-09 15:18

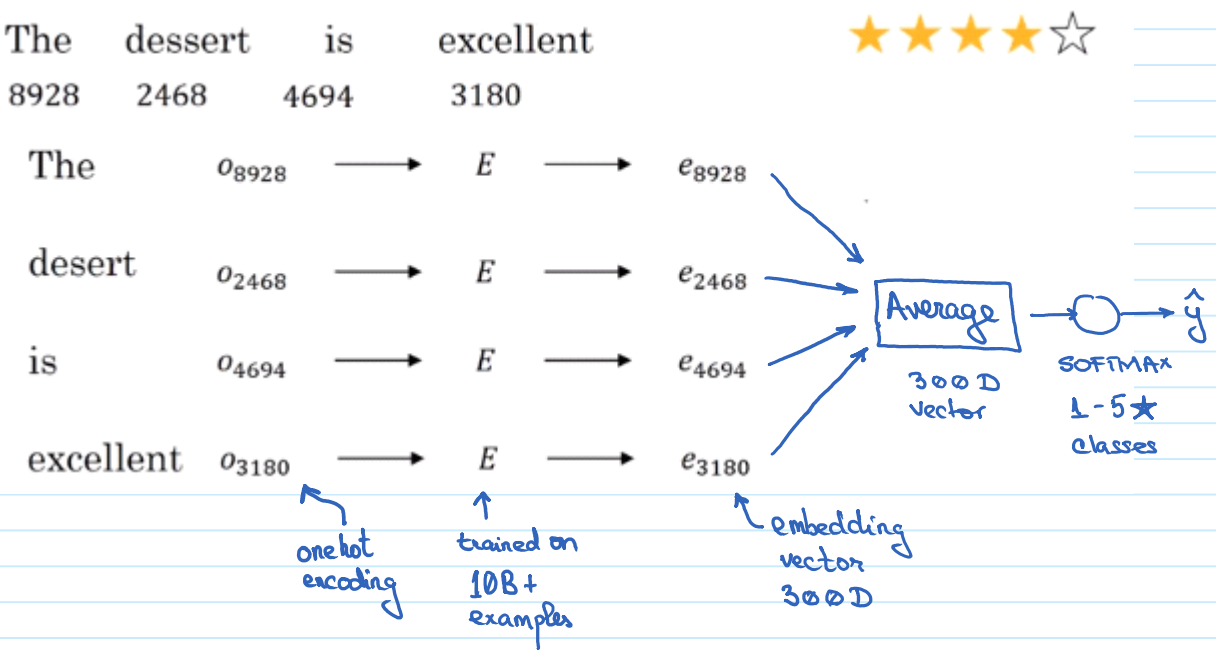
Training set size might be only 10k to 100k examples.

SIMPLE SENTIMENT CLASSIFICATION MODEL:

For the given training example below we use the standard 10k Vocabulary and a word Embedding (trained on a corpus of 10B+ words)

Screen clipping taken: 2018-09-09 15:23

The	dessert	is	excellent	★★★★☆



Using the AVERAGE layer we decouple the length of the sentence from the output. The SOFTMAX layer is agnostic to whether there were 10 or 100 words used as inputs.

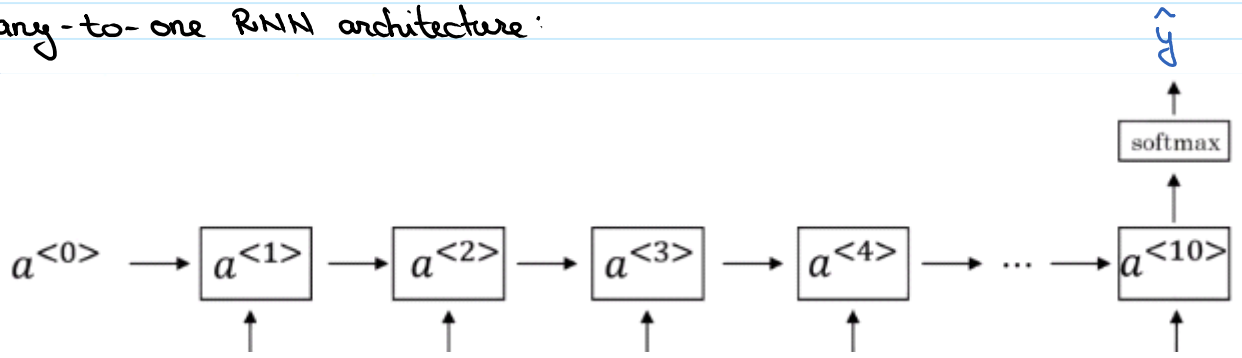
PROBLEM: This algorithm ignores word order. So even though the review below is negative because it uses a few "good" words it will be predicted as positive (we average the positive connotations of "good" in the predicted outcome).

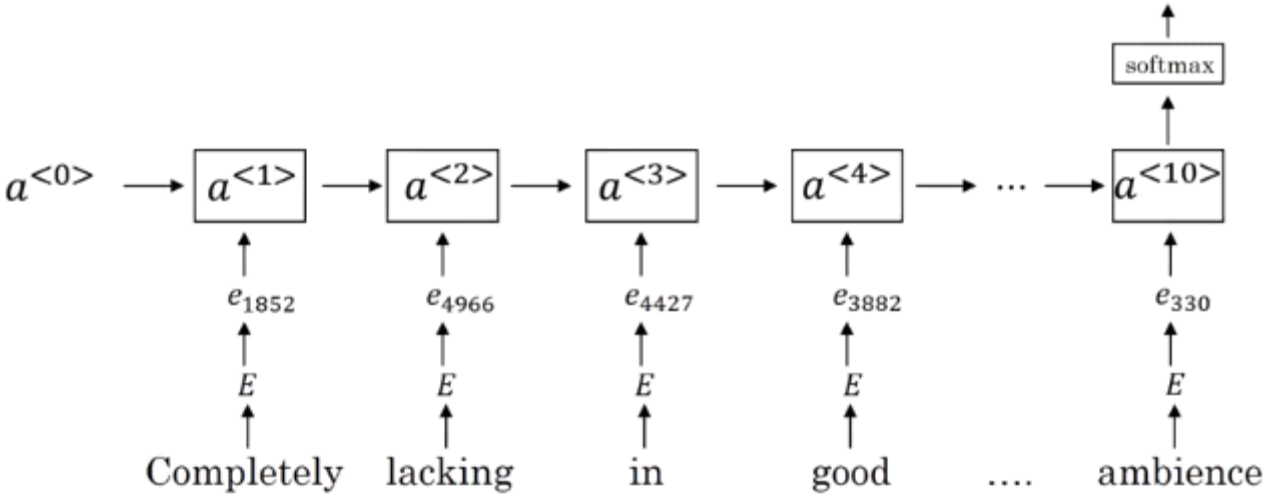
> "Completely lacking in good taste, good service, and good ambience."

Screen clipping taken: 2018-09-09 15:37

RNN MODEL FOR SENTIMENT CLASSIFICATION:

Many-to-one RNN architecture:





Screen clipping taken: 2018-09-09 15:40

This can properly account for word order.

→ it can determine "not good" has a negative connotation.

Using Word Embeddings that were trained on very large data sets improves the robustness of the RNN prediction (generalizes well).

For example if we have as test input:

"Completely absent of good ... ambiance"

and our 10k vocabulary doesn't contain the word "absent" the RNN will still perform well because it learned from the Word Embedding.

Debiasing word embeddings

Sunday, 09 September, 2018 15:49

Goal: As machine learning algorithms are used in more and more socially complex environments as crucial decision makers we want to ensure they are free of undesirable forms of bias (i.e. gender bias, ethnicity bias etc).

Learn about some approaches meant to diminish this type of bias in Word Embeddings.

Note: this is not related to neural network bias.

SOCIAL BIASES IN WORD EMBEDDINGS

[Bolukbasi et. al., 2016. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings]

Screen clipping taken: 2018-09-09 16:02

Neural Nets can learn analogies:

"Man" is to "Woman" as "King" is to "Queen".

The authors of the paper above found that NN predict

"Man" is to "Computer Programmer" as "Woman" is to "Homemaker"

Father : Doctor → Mother : Nurse

Ideally it would output

Man : Computer Programmer → Woman : Computer Programmer

Word embeddings can reflect gender, ethnicity, age, sexual orientation, and other biases of the text used to train the model.

Screen clipping taken: 2018-09-09 16:10

The NN social biases will affect very important decisions affecting our lives.

- job applications
- college admission
- loan approvals
- criminal sentencing

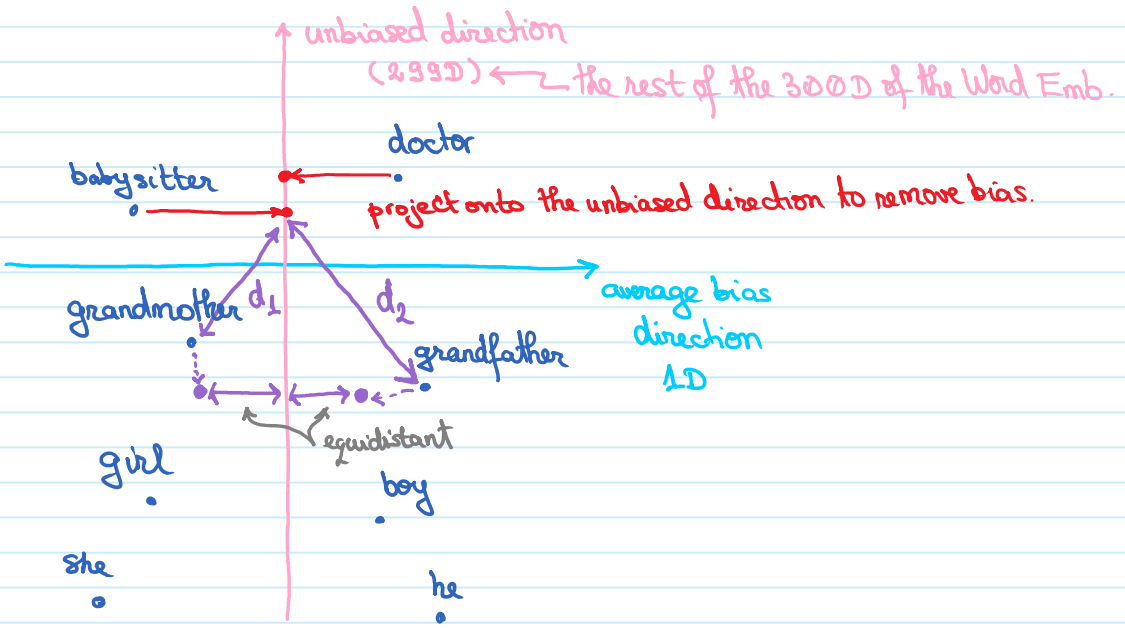
...

Word Embeddings pick-up the biases found in the corpus used to train them. (as it was written by humans)

We can more easily mitigate the biases in AI than the bias in the human population at large.

ADDRESSING SOCIAL BIAS IN WORD EMBEDDINGS

In this example we are focused on reducing gender bias (same idea applies to other types of bias).
Assume we have learned a Word Embedding like shown in the t-SNE projection below:



1. Identify bias direction

$e_{he} - e_{she}$
$e_{male} - e_{female}$
} average the Δ

In the original paper

- the bias direction can have more than 1D
- instead of average use Single Value Decomposition (SVD)
  (this is related to Principal Component Analysis Algorithm)

2. Neutralize bias direction

- For every word that is not definitional, project to remove bias.

words that intrinsically capture gender (ethnicity, etc)
i.e. "grandmother", "grandfather", "he", "she"

HOW DO WE DECIDE WHICH WORDS TO NEUTRALIZE?

	Neutralize
doctor	Y
grandmother	N
beard	N

To determine which words are "definitional" train a linear classifier to figure out which words are definitional (i.e. gender is part of the definition)

→ very few words in English are definitional.

3. Equalize pairs.

- We want the only difference in the embedding to be "gender" for some word pairs.

grandmother	grandfather
girl	boy
sorority	fraternity
⋮	

d_1: distance between "grandmother" & "babysitter"

d_2: distance between "grandfather" & "babysitter"

Having $d_1 < d_2$ reinforces negative stereotype.

This step moves "grandmother" & "grandfather" such that they are equidistant from the unbiased axis, in effect causing the two words in the pair to be at the same distance from babysitter ($d_1 = d_2$).

HOW DO WE DECIDE WHICH WORDS TO EQUALIZE?

Similar to the neutralization step there are a small number of pairs in the English language that need to be equalized

→ it is feasible to handpick the pairs we want to equalize.

Week 3 Sequence Models & Attention Mechanism

Wednesday, 29 August, 2018 21:56

Basic Model

Monday, 17 September, 2018 21:45

Goal: Learn about sequence-to-sequence models used in machine translation & speech recognition

SEQUENCE TO SEQUENCE MODEL

We want to translate a sentence from French to English.

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$ ← input sequence

Jane visite l'Afrique en septembre

→ Jane is visiting Africa in September.

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$

output sequence

Screen clipping taken: 2018-09-17 21:55

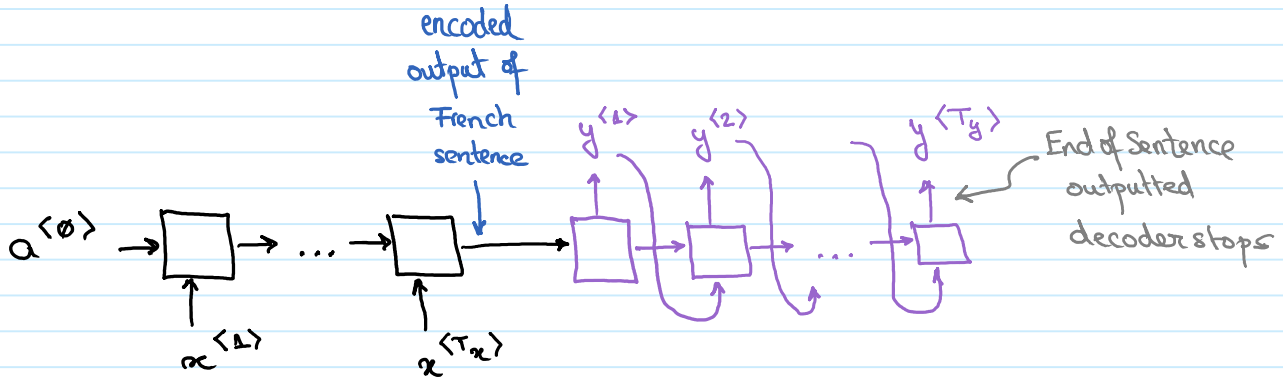
How do we train a NN to use sequence x as input and output y ?

As presented in the papers below:

[Sutskever et al., 2014. Sequence to sequence learning with neural networks]

[Cho et al., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation]

Screen clipping taken: 2018-09-17 21:53



(an RNN: GRU or LSTM)

outputs translation based on encoded output one word at a time

Surprisingly this encoder-decoder model works well given a large training set.

IMAGE CAPTIONING

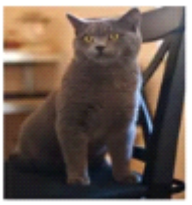
See work done independently by a few groups:

[Mao et. al., 2014. Deep captioning with multimodal recurrent neural networks]
[Vinyals et. al., 2014. Show and tell: Neural image caption generator]
[Karpathy and Fei Fei, 2015. Deep visual-semantic alignments for generating image descriptions]

Screen clipping taken: 2018-09-17 22:07

Given an image we want a classifier that outputs a descriptive sentence:

X

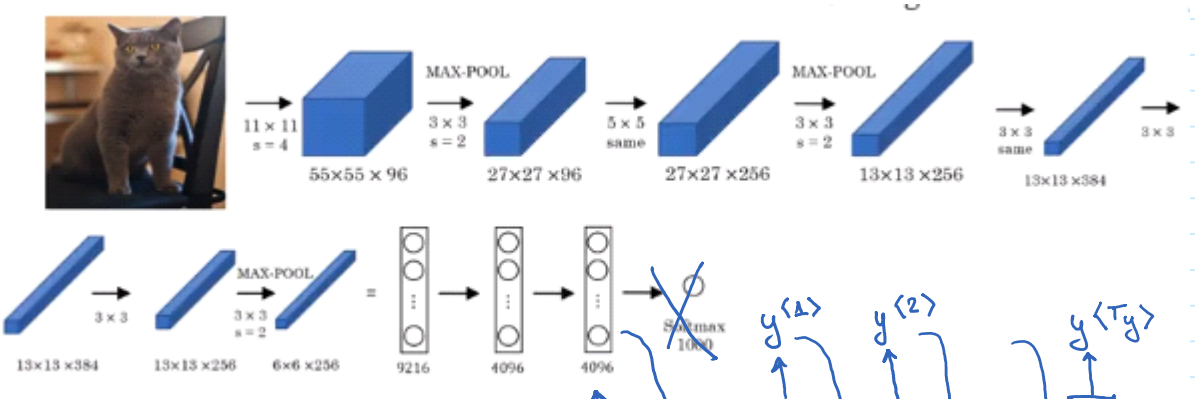


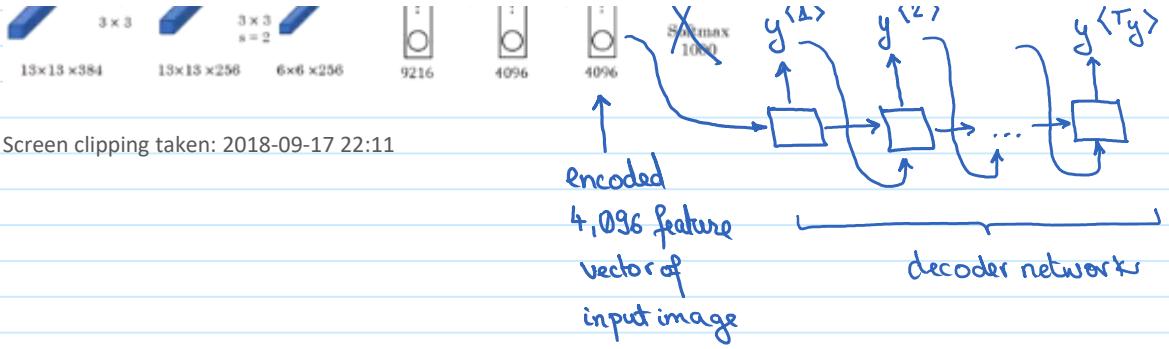
Y

$y^{<1>}$ A $y^{<2>}$ cat $y^{<3>}$ sitting $y^{<4>}$ on $y^{<5>}$ a $y^{<6>}$ chair

Screen clipping taken: 2018-09-17 22:09

We will use a ConvNet (i.e. AlexNet) to learn an encoding (i.e. a set of features from the input image)





Screen clipping taken: 2018-09-17 22:11

This method works well for creating captions for images as long as the captions are fairly short.

NUANCE:

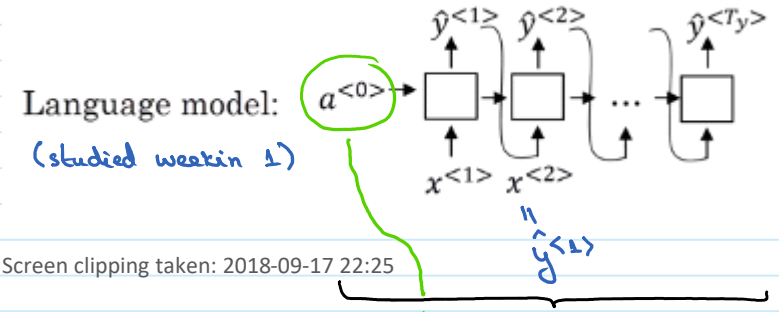
The difference between the sequence-to-sequence models and the language models we used before is that for the sequence-to-sequence models we don't want to generate a randomly chosen translation we want the most likely translation. See next lecture for details.

Picking the most likely sentence

Monday, 17 September, 2018 22:21

Goal: Learn more about the similarities & differences between the new sequence-to-sequence models studied this week and the language models studied in the previous weeks.

MACHINE TRANSLATION AS BUILDING A CONDITIONAL LANGUAGE MODEL



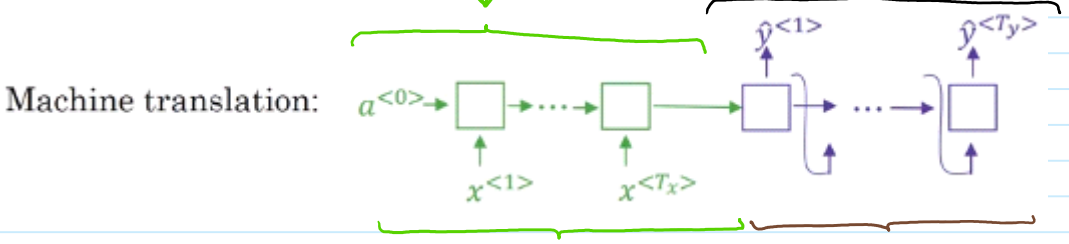
Screen clipping taken: 2018-09-17 22:25

This model is taused estimate probability the of a sentence (sequence of words)

$P(y^{<1>}, y^{<2>}, \ldots, y^{<T_y>})$

also (we can use it to generate new sentences)

Similar to decoder



Screen clipping taken: 2018-09-17 22:28

ENCODER NETWORK DECODER NETWORK

So the Machine Translation replaces the vector $a^{<0>}$ (a vector of all 0s) with a representation of the input sentence (the encoding) and use it to start the decoding network.

Andrew Ng calls this "Conditional language model" as it calculates the probability of different output English translations conditioned on the input French sentence.

$$P(y^{\langle 1 \rangle}, \dots y^{\langle T_y \rangle} \mid x^{\langle 1 \rangle} \dots x^{\langle T_x \rangle})$$

FINDING THE MOST LIKELY TRANSLATION

We don't want any English translation (i.e. sampled outputs at random).

We want the English sentence y that maximizes the conditional probability.

- → Jane is visiting Africa in September.
- → Jane is going to be visiting Africa in September.
- → In September, Jane will visit Africa.
- → Her African friend welcomed Jane in September.

} different samplings from the distribution $P(y^{\langle 1 \rangle} \dots y^{\langle T_y \rangle} \mid x)$

Screen clipping taken: 2018-09-18 22:06

We want y that maximize $P(y \mid x)$: $\underset{y^{\langle 1 \rangle}, \dots, y^{\langle T_y \rangle}}{\arg\max} \; P(y^{\langle 1 \rangle} \dots y^{\langle T_y \rangle} \mid x)$

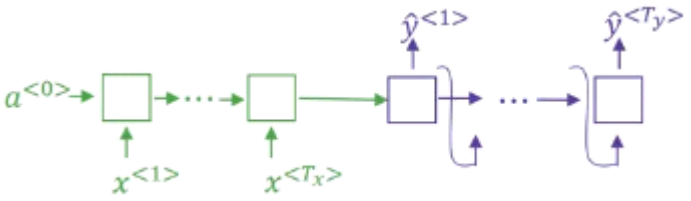
So to develop a machine learning algorithm we need to find algorithms that maximize the conditional probability.

→ the most common algorithm used is Beam Search

→ Why not use Greedy Search?

WHY NOT USE GREEDY SEARCH

Greedy Search is the algorithm that for each output at a given position in the sequence it chooses the most likely word given the conditional language model for that position



Screen clipping taken: 2018-09-18 22:13

What we would like is to choose the words in such a way that we select the sequence that has the highest probability overall (i.e. the joint probability).

The Greedy Algorithm will not provide the highest joint probability and is thus providing a non optimal solution

→ Jane is visiting Africa in September.

→ Jane is going to be visiting Africa in September.

Screen clipping taken: 2018-09-19 21:14

this is a very common word used after "is" so a Greedy Algo will choose it over "visiting" - even though "visiting" is a better choice.

i.e $P(\text{"Jane is going"} \mid x) > P(\text{"Jane is visiting"} \mid x)$

So it is not optimal to base our word picking strategy on choosing the best word at a time.

The number of possible sentences is extremely large so it is not feasible to search it exhaustively for the best joint probability

i.e. a vocabulary of 10K words used to construct 10 word sentences has $10K^{10}$ possible options to search through.

In this case we use an approximate search algorithm

→ it will try (not guaranteed) to find.

$$\arg\max_{y} P(y^{\langle 1 \rangle} \dots y^{\langle T_y \rangle} \mid x)$$

Beam search

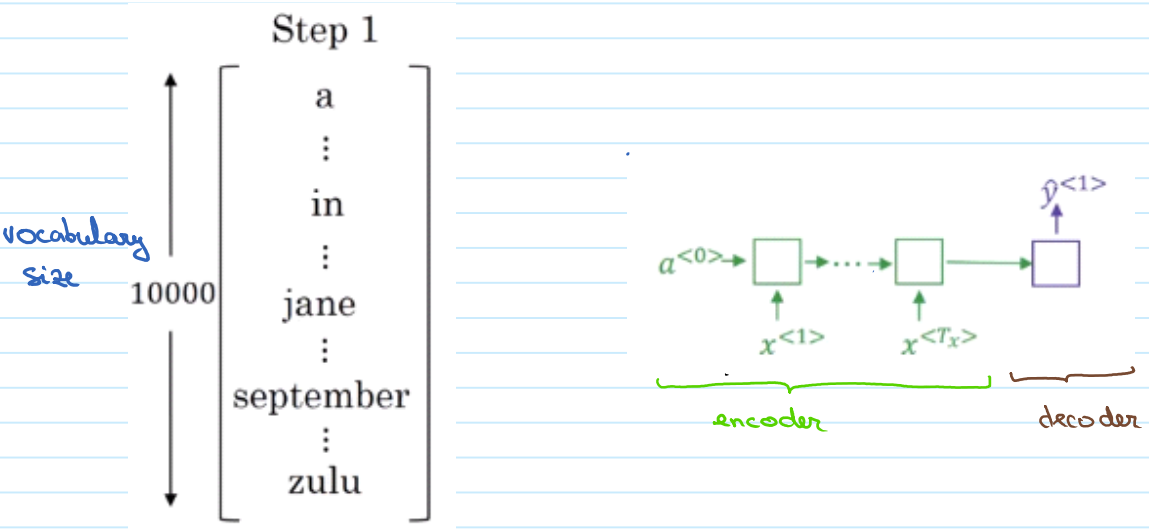
Wednesday, 19 September, 2018 21:32

Goal: Learn how Beam Search (an approximate search algorithm) works.

Beam search is applicable not only to machine translation but also to speech recognition (i.e. given an input audio clip we don't want to output a random transcript of the audio but instead we want to output the best transcript)

BEAM SEARCH ALGORITHM

First Beam Search tries to pick the first word of the translation

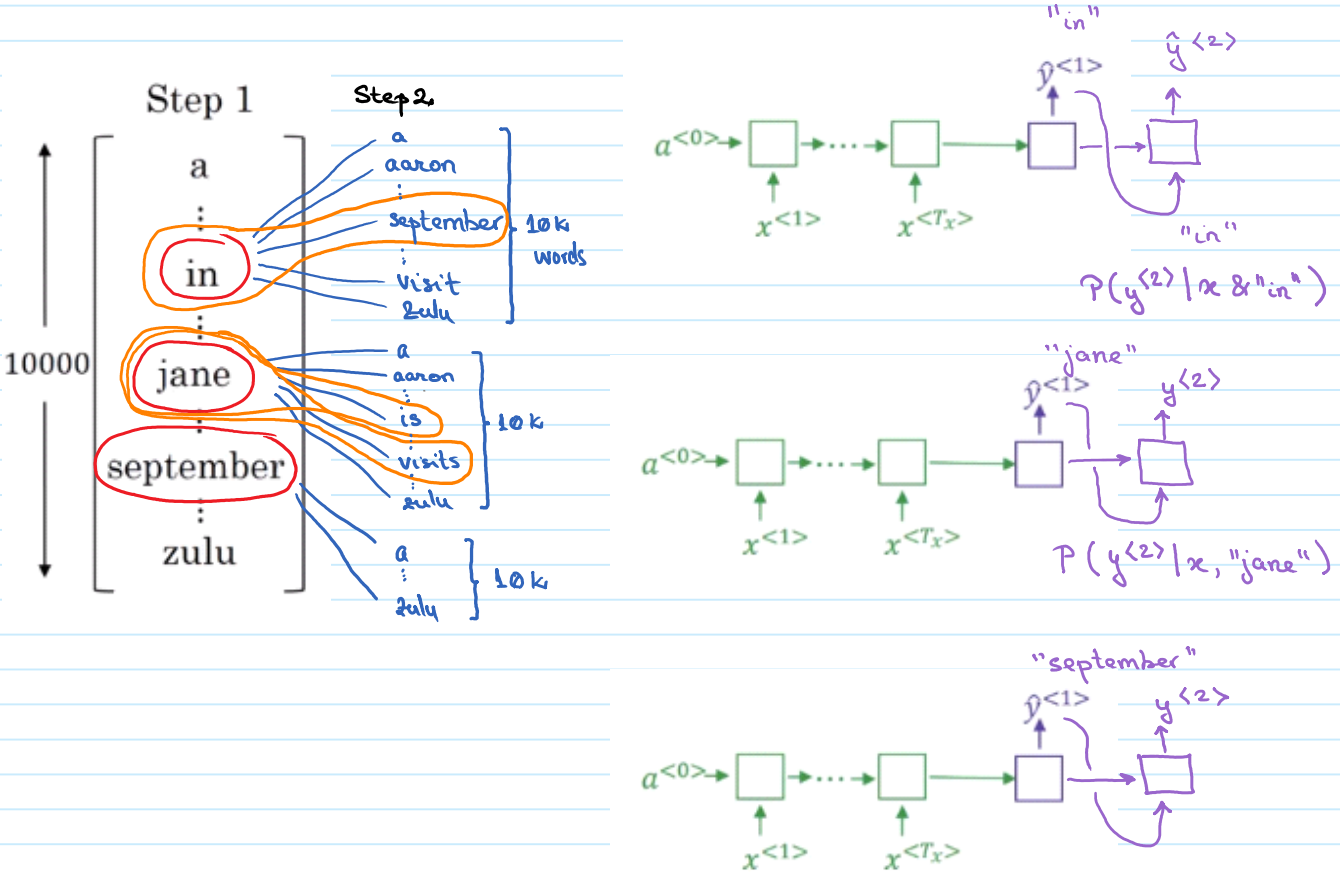


Screen clipping taken: 2018-09-19 21:42

So Greedy Search looks only at the probability of the current word when choosing which word to select.

Beam Search uses a parameter B (the beam width) which indicates that the selection should consider the top B most likely words when choosing each word.

Say B = 3 (in each step we use 3 words to determine what's the highest likelyhood)



What we want in the 2nd step is to find the pair of words $y^{<1>}$ & $y^{<2>}$ that has the highest probability (not just $y^{<2>}$ with highest probability)

$$P(y^{<1>}, y^{<2>} | x) = \underbrace{P(y^{<1>} | x)}_{1^{st} \text{ word probability}} \cdot \underbrace{P(y^{<2>} | x, y^{<1>})}_{2^{nd} \text{ word probability (given } 1^{st} \text{ word)}}$$

For each of the 3 words chosen in Step 1 we look at 10,000 words in Step 2. So in total we look at 30,000 possible words for $y^{<2>}$. Out of these 30k words we choose the top 3 words that have the highest $P(y^{<1>}, y^{<2>} | x)$

So after step 2 our top 3 choices for $y^{<1>}, y^{<2>}$ are:

"in september"

"jane is"

"jane visits"

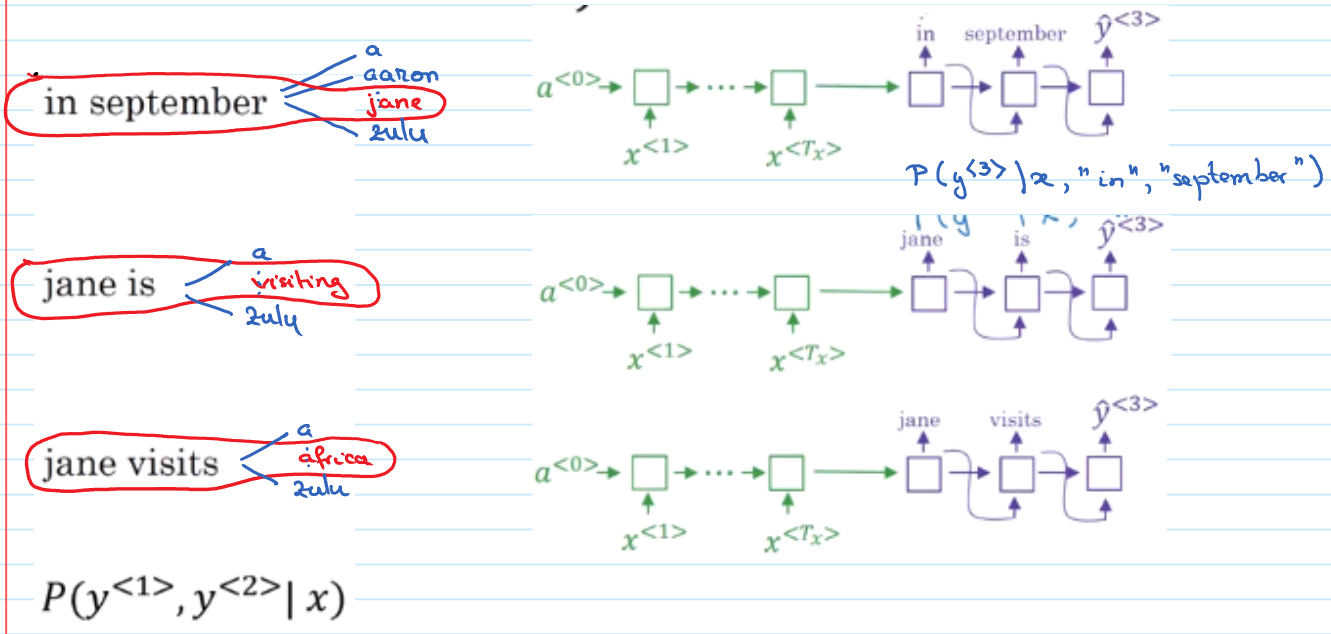
We then continue to Step 3.

Notice that after Step 2 the Beam Search algorithm has rejected "september" as a possible 1st word and we are left with either "in" or "jane" as 1st words.

→ though we are still keeping track of 3 pairs of words.

Because our Beam Width = 3 at every step we instantiate 3 sentence fragments to determine the step output (3 copies of the network, each evaluating all the 10k possible SOFTMAX outputs for that step)

BEAM SEARCH STEP 3



Screen clipping taken: 2018-09-20 22:33

So at the end of Step 3 we have the following 3 choices for $y^{<1>}, y^{<2>}, y^{<3>}$:

"in september jane"

"jane is visiting"

"jane visits africa"

By the end Beam Search determines that the most likely sentence is :

"jane visits africa in september <EOS>"

If Beam Width = 1 we revert to the Greedy Search algorithm.

Refinements to Beam Search

Sunday, 23 September, 2018 11:22

Goal: Learn about improvements to Beam Search

LENGTH NORMALIZATION

In Beam Search we want to choose $y^{\langle 1 \rangle} \dots y^{\langle T_y \rangle}$ so that we maximize the total probability

$$\arg\max_y \; P(y^{\langle 1 \rangle} \dots y^{\langle T_y \rangle} | x) \langle = \rangle$$

$$\arg\max_y \; P(y^{\langle 1 \rangle} | x) \, P(y^{\langle 2 \rangle} | x, y^{\langle 1 \rangle}) \dots P(y^{\langle T_y \rangle} | x, y^{\langle 1 \rangle} \dots y^{\langle T_y - 1 \rangle}) \; \langle = \rangle$$

$$\arg\max_y \prod_{t=1}^{T_y} P(y^{\langle t \rangle} | x, y^{\langle 1 \rangle}, \dots y^{\langle t-1 \rangle})$$

When we implement this each probability $P(y^{\langle t \rangle} | x, y^{\langle 1 \rangle}, \dots y^{\langle t-1 \rangle}) \in (0; 1)$

Often these probabilities are much less than 1, in fact they are so small that they cause under-flow (the value is so small the floating point representation on the computer can't store it accurately).

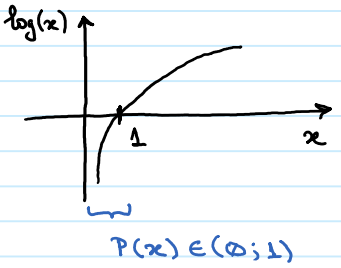
To get around this in practice we use logarithms of the probabilities product which gives us a more numerically stable algorithm that is less prone to rounding errors.

$$\arg\max_y \log \left[\prod_{t=1}^{T^{\langle y \rangle}} P(y^{\langle t \rangle} | x, y^{\langle 1 \rangle} \dots y^{\langle t-1 \rangle}) \right] =$$

$$\arg\max_y \sum_{t=1}^{T^{\langle y \rangle}} \log \left[P(y^{\langle t \rangle} | x, y^{\langle 1 \rangle} \dots y^{\langle t-1 \rangle}) \right]$$

Remember: $\log(a \cdot b) = \log(a) + \log(b)$

This works because $\log(x)$ is a strictly monotonically increasing function so:



$$\arg\max_x \left[f(x) \right] = \arg\max_x \left[\log f(x) \right] \quad \text{for } x > 0$$

and

$\log(x)$ is a very large negative number when $x \to 0$

If we have a very long sentence as output the probability of this sentence is low (because we are multiplying many subunitary values). So this objective function has a bias towards short output sentences (i.e. translations).

To get around this problem we can normalize the objective function:

$$\frac{1}{T_y} \sum_{t=1}^{T_y} \log P\left(y^{\langle t \rangle} | x, y^{\langle 1 \rangle}, \cdots, y^{\langle t-1 \rangle} \right)$$

which significantly reduces the penalty of outputting long sentences.

In practice we use a softer approach (somewhat of a hack, with no strong theoretical justification but with great practical results)

$$\frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log P\left(y^{\langle t \rangle} | x, y^{\langle 1 \rangle}, \cdots, y^{\langle t-1 \rangle} \right)$$

where $\alpha = 0.7$. If $\alpha = 1$ full normalization

$\alpha = 0$ no normalization.

and α is another hyperparameter.

SUMMARY OF RUNNING BEAM SEARCH WITH LENGTH NORMALIZATION

- As we run Beam Search we analyze many sentences of different lengths

  $T_y = 1, 2, \ldots 30$

  in this case we set a limit of 30 steps per output (words per sentence)

- For a Beam Width of 3 we will keep track of the top 3 most likely sentences analyzed above using the scoring function (objective function) below:

$$\frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log P(y^{\langle t \rangle} | x, y^{\langle 1 \rangle}, \ldots, y^{\langle t-1 \rangle})$$

  known as normalized log probability objective (or normalized log likelihood objective)

- At the end choose the sentence with the highest objective function and output it.

How To CHOOSE THE BEAM WIDTH B?

- PRO: The larger B is the more possibilities we are keeping track of and more likely to get the best combination of $y^{\langle 1 \rangle} \ldots y^{\langle T_y \rangle}$.

- CON: The larger B the more computationally expensive the algorithm is as we have to keep track of many more possibilities (i.e. 10k for each unit of B for a 10k vocabulary).

  → both time complexity and space complexity increase.

large B → better result, slower runtime

small B → worse result, faster runtime

In production a Beam Width of 10 is common, 100 is considered large

In research (highest quality papers) a B of 10000 is seen (even 30000)

This is application & domain dependent.

Unlike exact search algorithms like BFS (Breadth First Search) or DFS (Depth First Search), Beam Search runs faster but is not guaranteed to find exact maximum for $\arg\max_y P(y|x)$.

Screen clipping taken: 2018-09-23 12:57

Error analysis in Beam Search

Sunday, 23 September, 2018 13:06

Goal: Learn how to use Error Analysis on Beam Search to determine how well the beam width we chose works.

Beam Search is an approximate search algorithm, which means that it can sometimes output the wrong result. The problem is when the algorithm performs poorly it could be caused by:

- the Beam Search algorithm poor performance.
- the RNN model poor performance.

We will learn how Error Analysis interacts with Beam Search and how to tease the two causes above apart so we can focus our efforts on solving the right problem.

WORKING EXAMPLE:

Jane visite l'Afrique en septembre.

Human: Jane visits Africa in September. (y^*) → ground truth

Algorithm: Jane visited Africa last September. ($\hat{y}$) → inference

not a very good translation (it changed the meaning of the original French sentence)

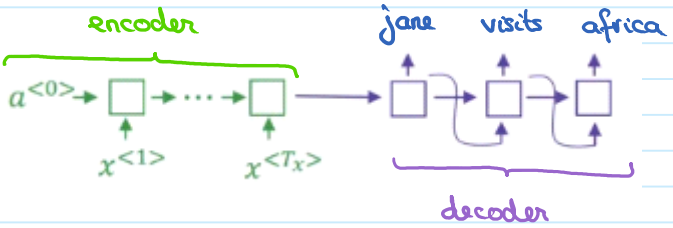
Screen clipping taken: 2018-09-23 13:25

Our model has two components:

- an RNN model
- a Beam Search model

and we want to determine to which one of these components we should attribute

the error.



Screen clipping taken: 2018-09-23 13:28

* Ignore upper/lower case.

Using our RNN model let's compute: $P(y^*|x)$ and $P(\hat{y}|x)$

We then compare the two results.

ERROR ANALYSIS ON BEAM SEARCH

Human: Jane visits Africa in September. (y^*)

Algorithm: Jane visited Africa last September. ($\hat{y}$)

Screen clipping taken: 2018-09-23 13:35

- Case 1: $P(y^*|x) > P(\hat{y}|x)$

  The Beam Search algorithm didn't find the better output y^*.
  Beam Search chose $\hat{y}$ but y^* has a higher $P(y^*|x)$.

  Conclusion: Beam Search is at fault

- Case 2: $P(y^*|x) \leq P(\hat{y}|x)$

  Beam Search actually found a better output than y^* given the RNN model, but this output was not very good.

According to the RNN $P(y^* | x) < P(\hat{y} | x)$ but y^* is a better translation than $\hat{y}$.

Conclusion : RNN is at fault.

Note : There are some length normalization subtelties so instead of using the raw probabilities $P(y^* | x)$ & $P(\hat{y} | x)$ we should use the normalized log likelihood objective from the previous lecture.

ERROR ANALYSIS PROCESS

Human	Algorithm	$P(y^* \mid x)$	$P(\hat{y} \mid x)$	At fault?
Jane visits Africa in September.	Jane visited Africa last September.	2×10^{-10}	1×10^{-10}	B
----------	----	----	----	R
----------	----	----	----	B
				R
				R

Screen clipping taken: 2018-09-23 13:49

- For each output we figure out as to whom we can attribute the error to.
- We then calculate the fraction of errors that are "due to" Beam Search vs RNN model
  - → only if the fraction of Beam Search errors is larger than RNN errors we should spend time to increase the Beam Width.

Bleu Score

Sunday, 23 September, 2018 14:34

Goal: Learn how to use BLEU score to measure accuracy for a machine translation system where there are multiple equally good outputs.

EVALUATING MACHINE TRANSLATION

[Papineni et. al., 2002. A method for automatic evaluation of machine translation] (quite a readable paper)

Screen clipping taken: 2018-09-23 14:41

We are given the input French sentence:

"Le chat est sur le tapis."

And multiple human references (ground truths)

Reference 1: The cat is on the mat.

Reference 2: There is a cat on the mat.

Screen clipping taken: 2018-09-23 14:53

BLEU : Bilingual Evaluation Understudy

- In the theater world an "understudy" is someone who learns the role of a more senior actor so they can replace them if necessary.
- The BLEU score in this case is the understudy to a human. It is used to evaluate every output of a machine translation - so that a human doesn't need to do it.

The BLEU score indicates how close the score of the translation generated by the RNN matches the score of any of the ground truths used (when passed through the RNN).

Intuition: Parse the machine translation sentence and check whether the types of words it generated appear in at least one of the human generated translations (ground truths).

→ ground truths would be provided as part of the dev set or test set.

So given:

French: Le chat est sur le tapis.

Ground Truth

Reference 1: The cat is on the mat.

Reference 2: There is a cat on the mat.

Screen clipping taken: 2018-09-23 15:09

And the machine translation outputs

MT output: the the the the the the the. (a very poor translation)

7 words

Using Precision (the fraction of the words in the MT output also appear in the References) to evaluate the output is not very informative in this case.

Precision: $\frac{7}{7}$

We will use Modified Precision instead. Modified Precision gives each word credit up to the maximum amount of times it appears in the reference sentences.

French: Le chat est sur le tapis.

Reference 1: The cat is on the mat. (2)

Reference 2: There is a cat on the mat. (1)

we choose the max #

so the word "the" gets credit up to 2 credits.

MT output: the the the the the the the.

Modified Precision: $\frac{2}{7}$ ← max count of "the" in references: $Count_{clip}$("the")

↑ count of "the" in MT output: Count("the")

BLEU SCORE ON BIGRAMS

So far we have looked at single words in isolation (unigrams). When using BLEU Score we also want to look at pairs of words appearing next to each other (bigrams).

To compute the BLEU Score we will take into account both bigrams and unigrams as well as longer sequences of words (like trigrams)

Example: Reference 1: The cat is on the mat.

Reference 2: There is a cat on the mat.

MT output: The cat the cat on the mat. ← slightly better translation

↑ bigrams in the MT output

Screen clipping taken: 2018-09-23 15:30

	Count	$Count_{clip}$
the cat	2	1
cat the	1	0
cat on	1	1
on the	1	1
the mat	1	1
	6	4

Modified Precision on bigrams = $\frac{Count_{clip}}{Count} = \frac{4}{6}$

Screen clipping taken: 2018-09-23 15:36

FORMALIZED BLEU SCORE ON GRAMS

Example: Reference 1: The cat is on the mat.

Reference 2: There is a cat on the mat.

MT output: The cat the cat on the mat. $\hat{y}$

Screen clipping taken: 2018-09-23 15:43

- Unigram :

precision

$$P_1 = \frac{\sum_{\text{unigram} \in \hat{y}} \text{Count}_{\text{clip}}(\text{unigram})}{\sum_{\text{unigram} \in \hat{y}} \text{Count}(\text{unigram})}$$

unigram

- N-gram :

$$P_n = \frac{\sum_{\text{n-gram} \in \hat{y}} \text{Count}_{\text{clip}}(\text{unigram})}{\sum_{\text{n-gram} \in \hat{y}} \text{Count}(\text{unigram})}$$

n-gram

If the MT output is identical to either Ref 1 or Ref 2 then

$P_1 = P_2 = \ldots = P_n = 1.0$.

sometimes this is possible even when the MT output is different than Ref 1 or Ref 2.

BLEU SCORE DETAILS

p_n = Bleu score on n-grams only $\quad P_1, P_2, P_3, P_4$

Combined Bleu score: $\quad BP \cdot e^{\left(\frac{1}{4}\sum_{n=1}^{4} P_n\right)}$

Screen clipping taken: 2018-09-23 15:50

exp is a strictly monotonically

↑ exp is a strictly monotonically increasing function

BP = brevity penalty

→ it's easier for short outputs to match the References words and get a high precision score. But we don't want to bias the score towards translations that are very short so we penalize translation outputs that are too short.

$$BP = \begin{cases} 1 & \text{if MT\_output\_length} > \text{reference\_output\_length} \\ e^{\left(1 - \frac{\text{MT\_output\_length}}{\text{reference\_output\_length}}\right)} & \text{otherwise} \end{cases}$$

- The BLEU score was revolutionary in machine translation because it provided a pretty good (not perfect) single real number evaluation metric to automatically determine if a machine generated text has similar meaning as a reference piece of text generated by humans. It was then used to accelerate the progress of the entire field.
- But the BLEU score is also used in image captioning system
- It is not used in speech recognition as there we have an exact representation of the ground truth.

In practice few people implement the BLEU score → everyone uses downloaded open source implementations.

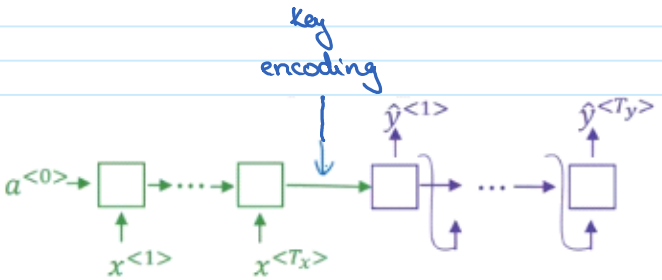
Attention Model Intuition

Sunday, 23 September, 2018 16:34

Goal: Learn about the Attention Model - a model that improves on the encoder - decoder model used with RNNs.

THE PROBLEM OF LONG SEQUENCES:

Given a very long sentence to translate we are asking the RNN to encode all of it in the activations passed to the decoder segment and then the decoder has to generate the entire translation from this one encoding.

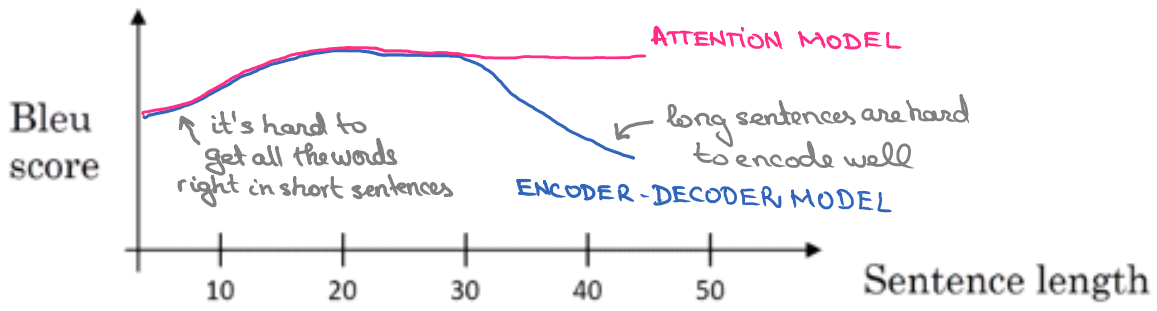


Jane s'est rendue en Afrique en septembre dernier, a apprécié la culture et a rencontré beaucoup de gens merveilleux; elle est revenue en parlant comment son voyage était merveilleux, et elle me tente d'y aller aussi.

Jane went to Africa last September, and enjoyed the culture and met many wonderful people; she came back raving about how wonderful her trip was, and is tempting me to go too.

Screen clipping taken: 2018-09-23 16:39

This is increasingly difficult as the sentence length increases and because of this the encoder - decoder RNN performance decays for longer sentences



Screen clipping taken: 2018-09-23 16:44

The way a human translates the sentence is to break it up into sub sentences and

translate those part by part and build up the translated sentence.

With attention models we work the sentence one part at a time. As this method doesn't push the ability of the network to encode (memorize) long sequences beyond it's optimum we will maintain good performance even for longer sentences.

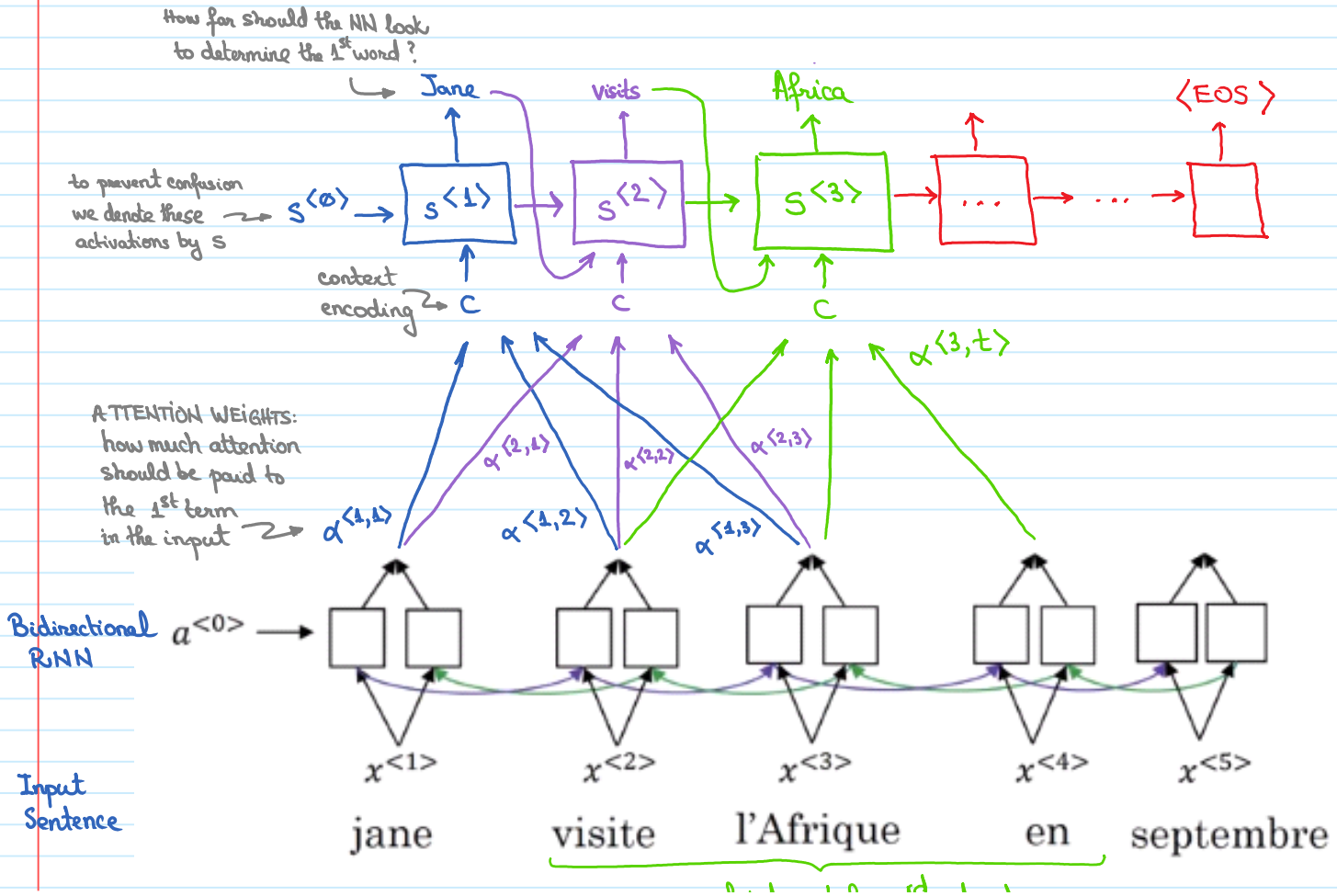
ATTENTION MODEL INTUITION

[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]

↳ this is one of the seminal papers in the deep learning literature.

Screen clipping taken: 2018-09-23 17:00

Even though the attention model was developed for machine translation it has many other applications.



Sentence

jane visite l'Afrique en septembre

area of interest for 3rd output determined by c

Screen clipping taken: 2018-09-23 17:03

- The goal of the context is to define which part of the sentence needs to be used to translate the associated word.

- The attention model is computing a set of attention weights.

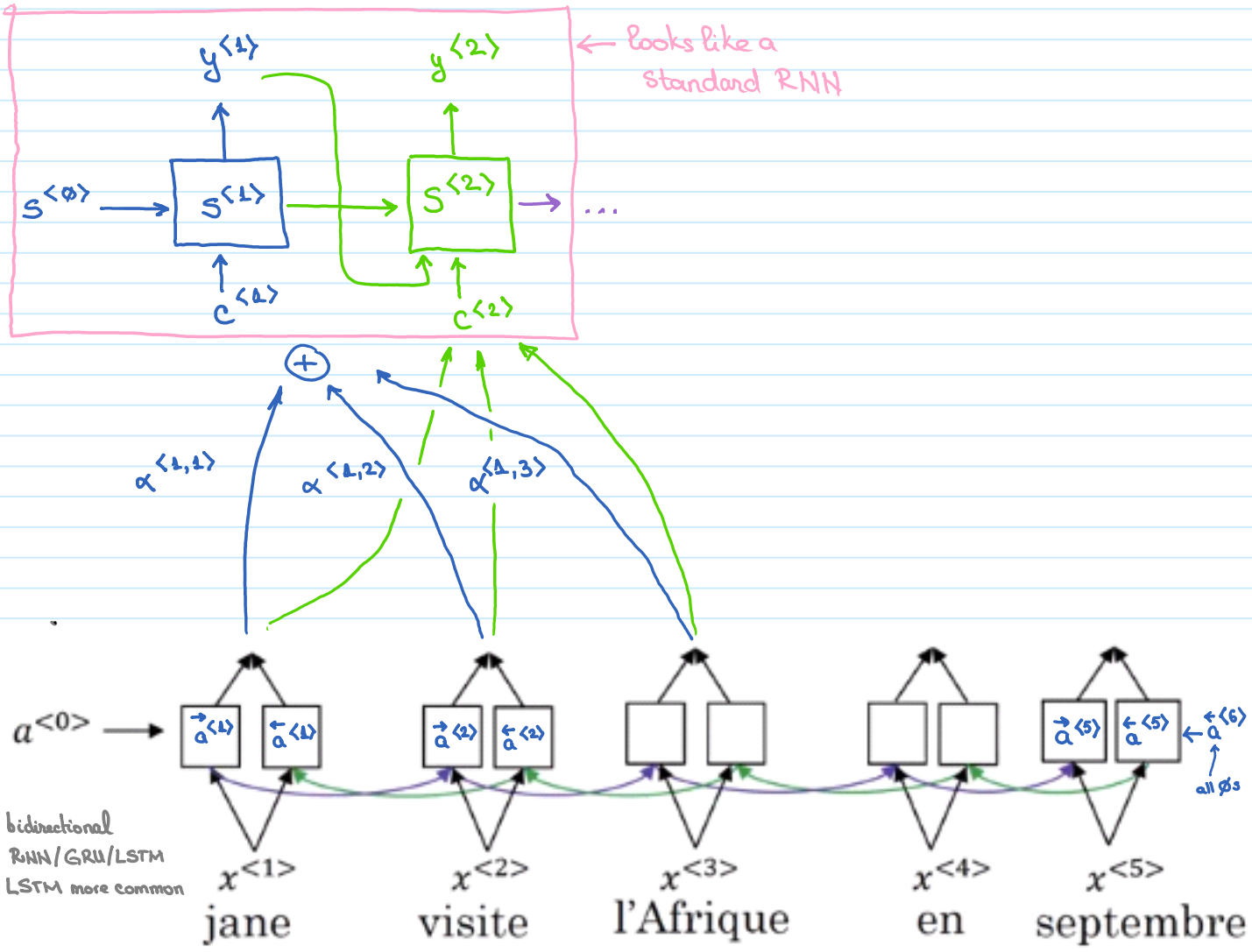
- The attention weight $\alpha^{\langle u,t \rangle}$ tells us when we try to generate the u^{th} English word how much should we be paying attention to the t^{th} French word.
  → this limits the inputs for the u^{th} output to within a local window of inputs.

- A given attention weight i.e. $\alpha^{\langle 3,t \rangle}$ (how much to pay attention to a specific word from the input French sentence) depends on:
  - the activations of the bidirectional RNN at step t $\vec{a}^{\langle t \rangle}$ & $\overleftarrow{a}^{\langle t \rangle}$ (both forward & backwards)
  - previous step state $s^{\langle 2 \rangle}$

Attention Model

Sunday, 23 September, 2018 17:56

Goal : Formalize the intuition we developed for Attention models



Screen clipping taken: 2018-09-23 18:15

$\alpha^{<u,t>}$: amount of "attention" $y^{<u>}$ (u^{th} output word) should pay to $\bar{a}^{<t>}$ (activations of t^{th} input word)

We use $\bar{a}^{<t>}$ to denote both forward & reverse activations for step t

$$\bar{a}^{<t>} = (\vec{a}^{<t>}, \overleftarrow{a}^{<t>})$$

The attention weights satisfy :

- $\alpha^{\langle u,t \rangle} \geq 0$

- $\sum_t \alpha^{\langle u,t \rangle} = 1$

The context at step 1 of the output:

$$c^{\langle 1 \rangle} = \sum_t \alpha^{\langle 1,t \rangle} \bar{a}^{\langle t \rangle}$$

and at step 2 of the output:

$$c^{\langle 2 \rangle} = \sum_t \alpha^{\langle 2,t \rangle} \bar{a}^{\langle t \rangle}$$

COMPUTING ATTENTION WEIGHTS $\alpha^{\langle u,t \rangle}$

[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]
[Xu et. al., 2015. Show attention and tell: neural image caption generation with visual attention]

Screen clipping taken: 2018-09-23 18:38

$$\alpha^{\langle u,t \rangle} = \frac{e^{e^{\langle u,t \rangle}}}{\sum_{t=1}^{T_x} e^{e^{\langle u,t \rangle}}}$$

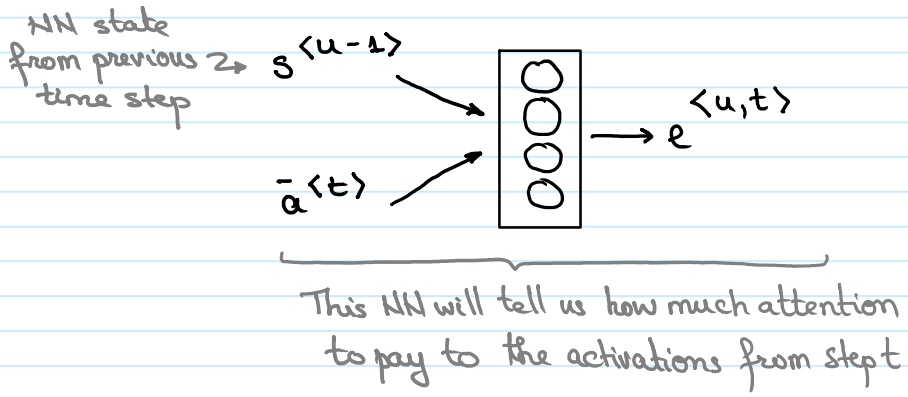
We are essentially using a SOFTMAX to ensure the weights sum to 1.

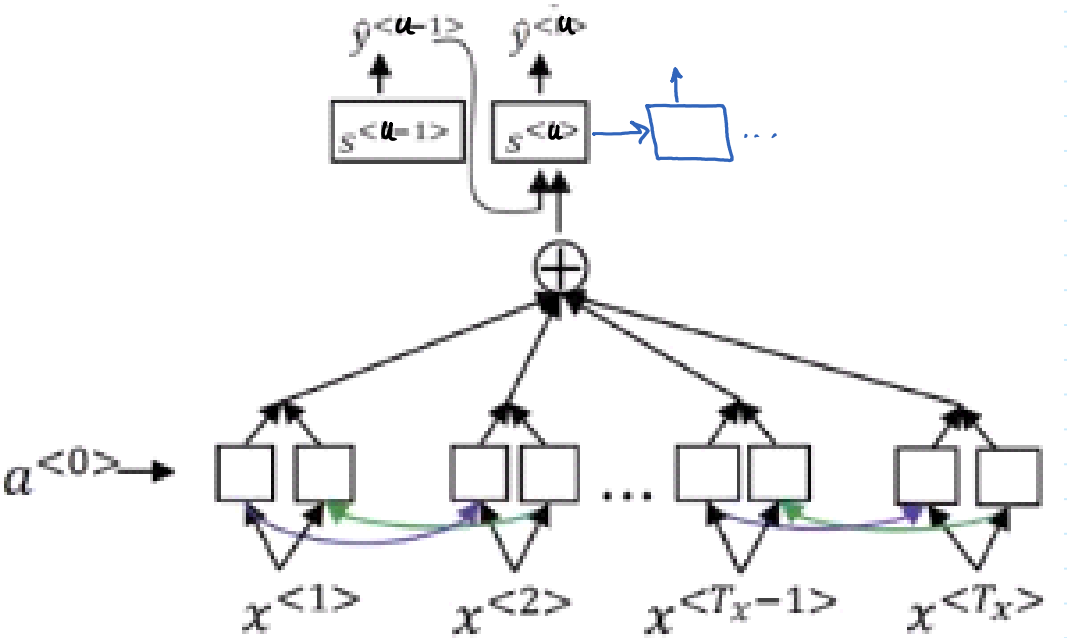
How do we compute the $e^{\langle u,t \rangle}$ factors?

→ use a small one-hidden layer NN with $s^{\langle u-1 \rangle}$ and $\bar{a}^{\langle t \rangle}$ as inputs

- The intuition is that $e^{\langle u,t \rangle}$ (and by extension $\alpha^{\langle u,t \rangle}$) should depend on $s^{\langle u-1 \rangle}$ and $a^{\langle t \rangle}$ but we don't know what the function is.

So we build a small NN and trust back-prop (& gradient descent) to learn the right function.





Screen clipping taken: 2018-09-23 18:52

One downside to this Attention Model algorithm is that it takes quadratic time. So for:

T_x words in input

T_y words in output

there are $T_x \times T_y$ attention weights $\alpha^{<t,t'>}$

In machine translation applications where neither T_x nor T_y are large this is acceptable.

This algorithm works well for both machine translation and image caption (where we look at parts of the image at one time).

ATTENTION EXAMPLE

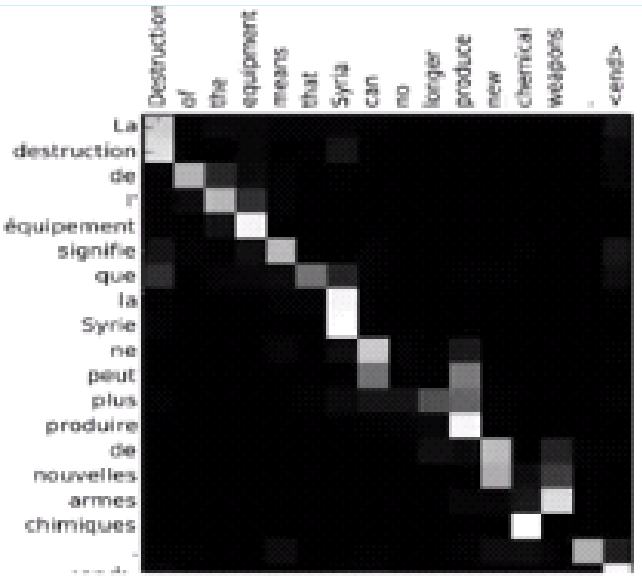
The Date Normalization Problem:

Input		Output	
July 20th 1969	⟶	1969 – 07 – 20	Moon landing
23 April, 1564	⟶	1564 – 04 – 23	Shakespeare's BD

Screen clipping taken: 2018-09-23 19:06

We can also visualize the attention weights:

Visualization of $\alpha^{<t,t'>}$:



Screen clipping taken: 2018-09-23 19:07

This indicates that the Attention Model seems to be paying attention to the correct inputs when generating an output.

Speech recognition

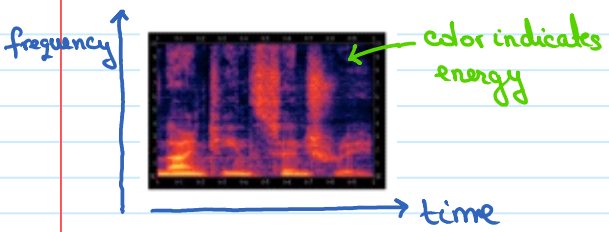
Sunday, 23 September, 2018 19:13

Goal: Learn how sequence-to-sequence models are applied to audio data.

SPEECH RECOGNITION PROBLEM:

Screen clipping taken: 2018-09-23 19:19

generate spectrogram / filtered bank outputs



Screen clipping taken: 2018-09-23 19:21

Back in the day speech recognition was performed using hand built phonemes (basic units of sound)

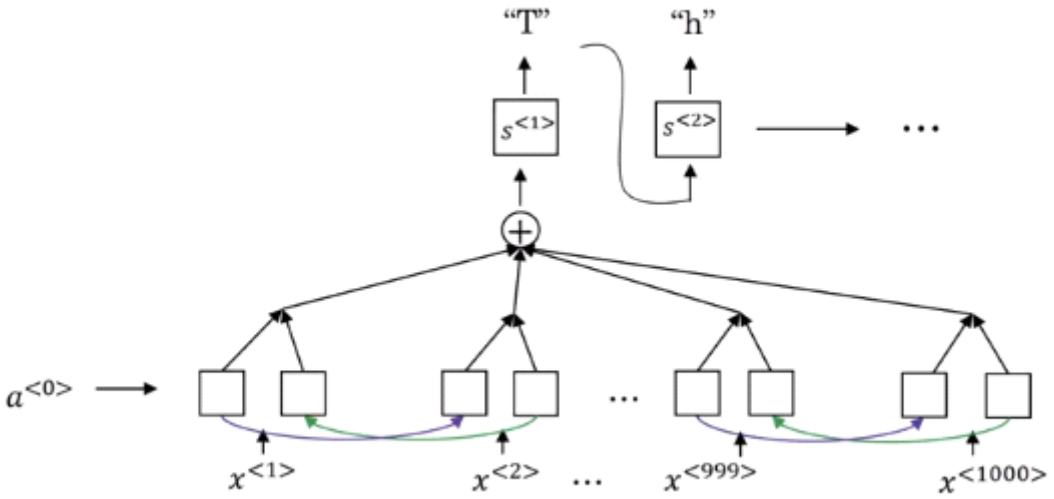
phonemes: de kwik brawn

When using end-to-end deep learning we are learning that hand engineered representations (like phonemes) are no longer needed.

→ this was made possible by access to large datasets

- 300h to 3000h of audio transcript is considered reasonable for research needs.
- commercial systems are trained on 100k hours of transcription

ATTENTION MODEL FOR SPEECH RECOGNITION:



Screen clipping taken: 2018-09-23 19:32

CTC COST FOR SPEECH RECOGNITION

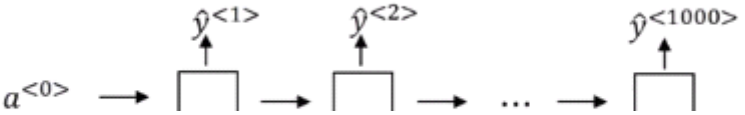
CTC: Connectionist temporal classification.

[Graves et al., 2006. Connectionist Temporal Classification: Labeling unsegmented sequence data with recurrent neural networks]

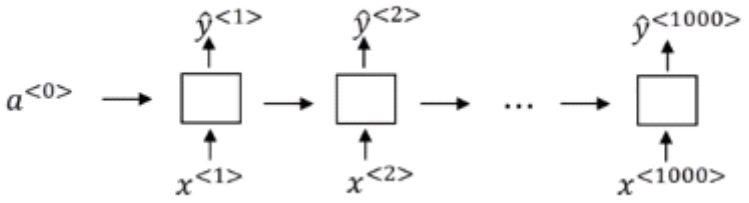
Screen clipping taken: 2018-09-23 19:33

Say we have an audio clip in which someone says:

"the quick brown fox"



"the quick brown fox"



Screen clipping taken: 2018-09-23 19:34

- Equal number of inputs & outputs $T_x = T_y = 1k$
- For ease of explanation we use a unidirectional RNN
- In practice this would be a bidirectional LSTM or GRU that's deeper

Notice the large number of time steps.

→ in speech recognition the number of input time steps (T_x) is much larger than the number of outputs (T_y)

- 10 sec audio at 100 Hz ⇒ 1000 inputs but the output will not have 1000 characters.

WHAT DO WE DO?

The CTC allows the RNN to generate an output as follows

ttt_h_eee ___␣___ 222__

t h e

Special character "blank character" NOT SPACE

Space character

which is considered a correct output for the first part of "the quick brown fox" ↑ 19 characters

Basic rule: collapse repeated characters not separated by "blank"

Screen clipping taken: 2018-09-23 19:44

So it can represent 19 characters using 1000 outputs.

To build a production speech recognition system requires vast amounts of data and we won't attempt it. Instead we'll build a trigger word detector.

Trigger word detection

Sunday, 23 September, 2018 19:48

Goal: Learn how to build a Trigger Word Detection System.

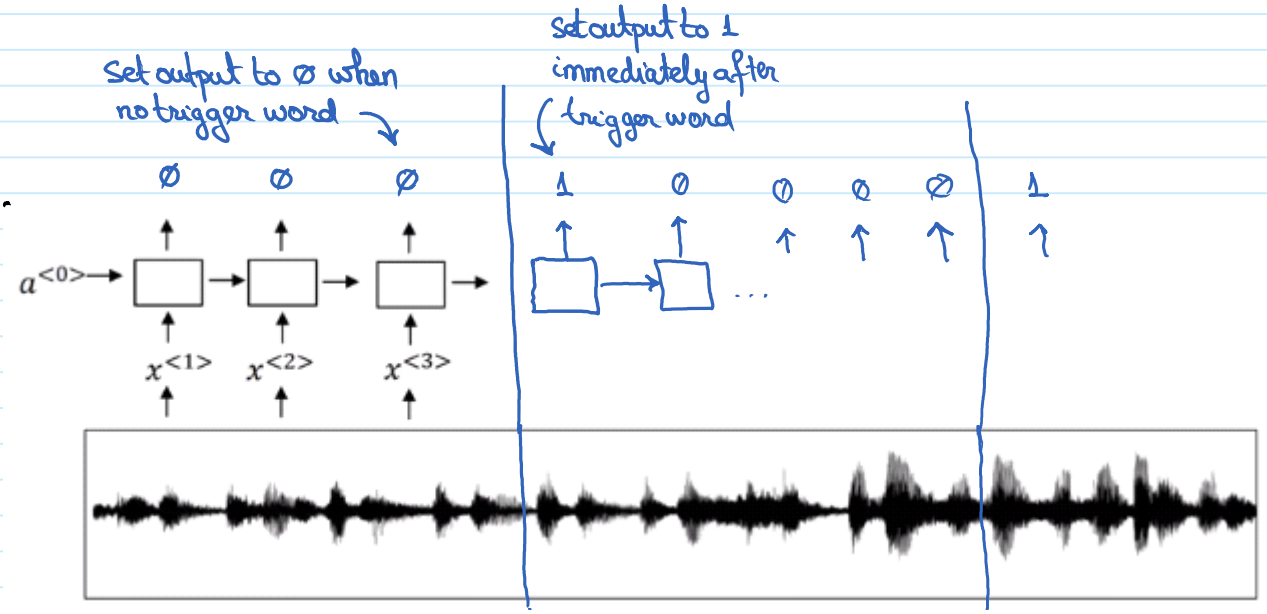
WHAT is TRIGGER WORD DETECTION

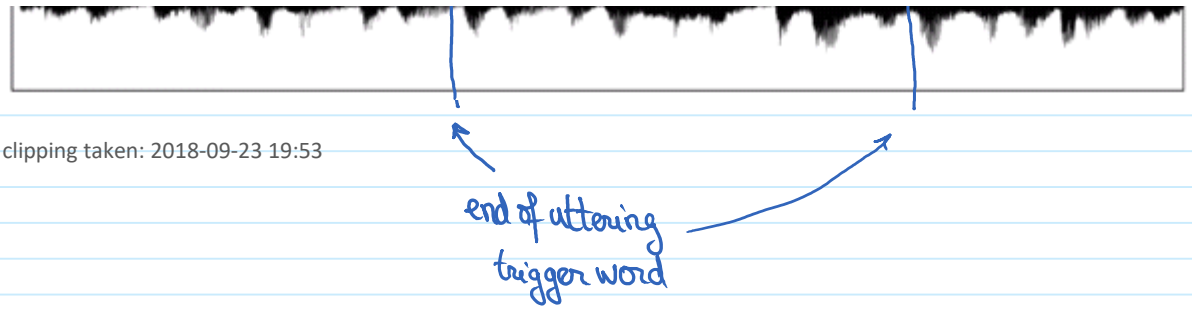
Screen clipping taken: 2018-09-23 19:50

TRIGGER WORD DETECTION ALGORITHM

The research is still evolving so there is no consensus on best approach.

This is one example.





Screen clipping taken: 2018-09-23 19:53

This approach causes an unbalanced training set (with many 0s and few 1s)

→ a hack would be to output a few more 1s instead of just a single 1.

Conclusion

Sunday, 23 September, 2018 19:59

Specialization outline

1. Neural Networks and Deep Learning
2. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization
3. Structuring Machine Learning Projects
4. Convolutional Neural Networks
5. Sequence Models

Screen clipping taken: 2018-09-23 20:00

DEEP LEARNING IS A SUPER POWER
(use it responsibly)

Screen clipping taken: 2018-09-23 20:01