COURSE 5
WEEK 1

SEQUENCE MODELS

Recurrent Neural Networks

Why Sequence Models

Examples of Sequence Data

Speech Recognition		"The quick brown fox jumped over the lazy dog" → Both input and output are sequence data
Music Generation	∅ can be nothing or integer	 → Sequence data
Sentiment Classification	"There's nothing to like in this movie"	
DNA Sequence Analysis	AGCCCTG...	AGCCC...
Machine Translation	Voluez-vous chanter avec moi?	Do you want to sing with me?
Video Activity Recognition		Running
Name Entity Recognition	Yesterday, Harry Potter met Hermoni Granger	Harry Potter Hermoni Granger

Notation

Motivating Example:

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

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$ $x^{<t>}$ $x^{<9>}$

y: 1 1 0 1 1 0 0 0 0

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

$X^{(i)}$ → i^{th} training example

$X^{(i)<t>}$ → i^{th} training example t^{th} element

$T_x^{(i)}$ → length of input in example i

$T_y^{(i)}$ → " " output in example i

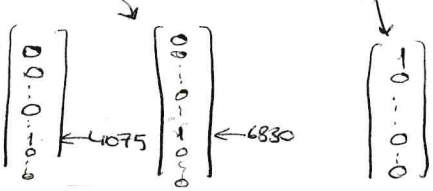
Representing words:

Vocabulary

a	1
aaron	2
and	367
harry	4075
potter	6830
zulu	10000

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

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



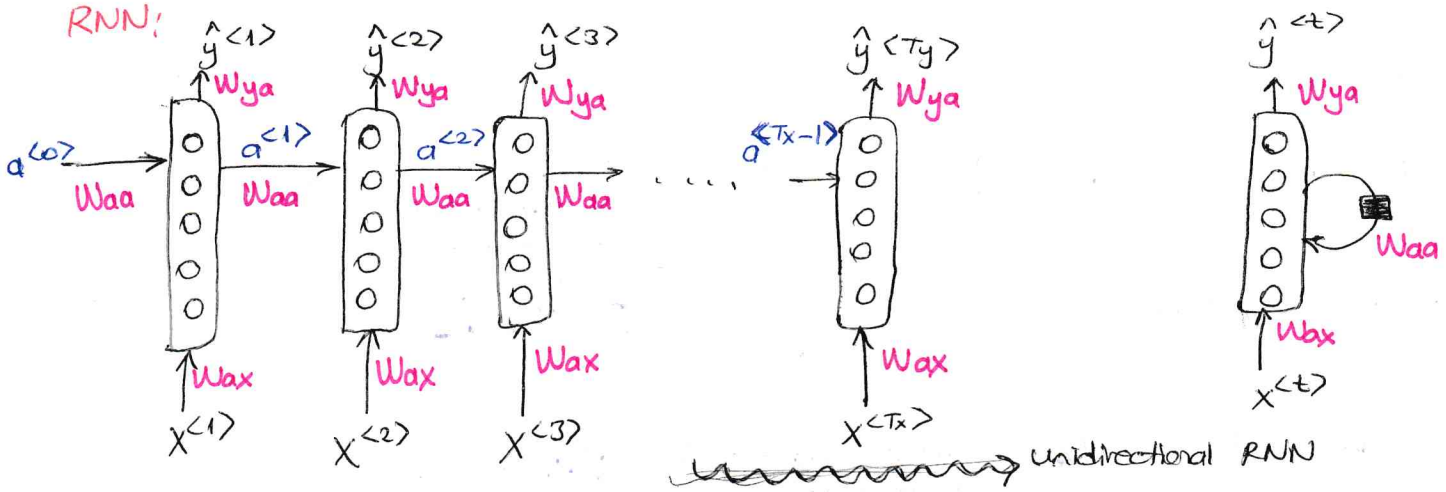
one-hot representation

Recurrent Neural Network Model

Why not a standart network?

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

RNN:



He said, "Teddy Roosevelt was a great President"
He said, "Teddy bears are on sale!"

} In order to detect the names, it necessitates bidirectional RNN

$a^{<0>} = \vec{0}$

$$a^{<1>} = g(W_{aa} \cdot a^{<0>} + W_{ax} \cdot x^{<1>} + b_a) \leftarrow \tanh / \text{ReLU}$$
$$\hat{y}^{<1>} = g_2(W_{ya} \cdot a^{<1>} + b_y) \leftarrow \text{sigmoid / softmax}$$

↳ more common

$$a^{<t>} = g(W_{aa} \cdot a^{<t-1>} + W_{ax} \cdot x^{<t>} + b_a)$$
$$\hat{y}^{<t>} = g_2(W_{ya} \cdot a^{<t>} + b_y)$$

Simplified RNN:

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

For example:

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

W_{aa}: (100, 100), W_{ax}: (100, 10000)

$$\underbrace{\begin{bmatrix} W_{aa} & W_{ax} \end{bmatrix}}_{100 \times 10100} = W_a \quad (100, 10100)$$

$$\begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = \begin{bmatrix} a^{<t-1>} \\ \hline x^{<t>} \end{bmatrix} \begin{matrix} \updownarrow 100 \\ \updownarrow 10000 \end{matrix} \quad \updownarrow 10100$$

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

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

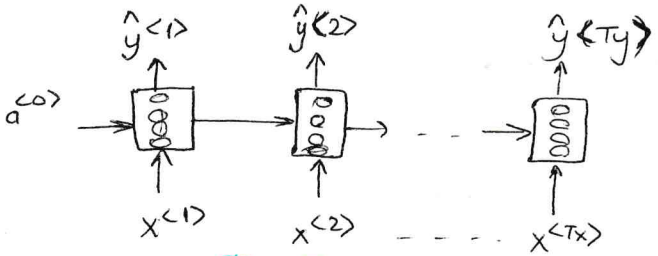
Backpropagation Through Time

$$L^{\langle t \rangle}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle}) = -y^{\langle t \rangle} \log \hat{y}^{\langle t \rangle} - (1-y^{\langle t \rangle}) \cdot \log(1-\hat{y}^{\langle t \rangle})$$

$$L(\hat{y}, y) = \sum_{t=1}^{T_y} L^{\langle t \rangle}(\hat{y}^{\langle t \rangle}, y^{\langle t \rangle})$$

Different Types of RNNs

The presentation in this video was inspired by a blog post by Andrej Karpathy, titled, The Unreasonable Effectiveness of Recurrent Neural Networks.



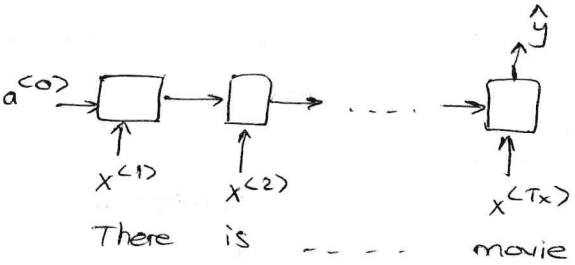
$T_x = T_y$

Many-to-many

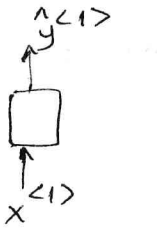
Sentiment classification

x = text

y = 0/1 ... 1...5



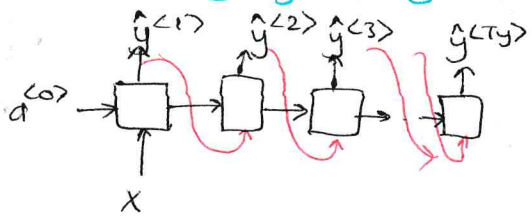
Many-to-one architecture



one-to-one

Music Generation

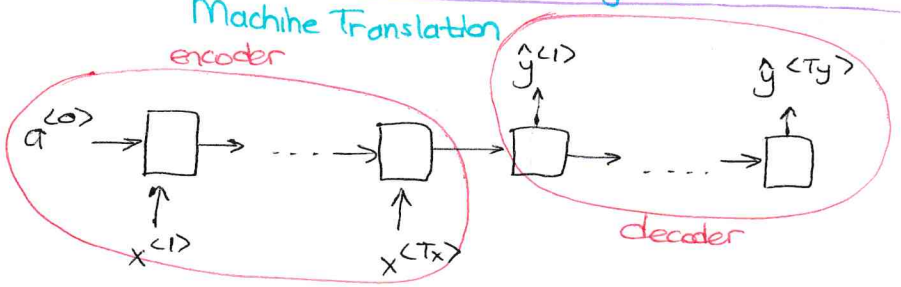
$x \rightarrow y^{\langle 1 \rangle} y^{\langle 2 \rangle} \dots y^{\langle T_y \rangle}$



One-to-many

~~$x = \emptyset$~~ (x can may be)

Machine Translation



Many-to-many

Language Model and Sequence Generation

What is language modelling?

Speech Recognition

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

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

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

Language modelling with an RNN

Training set : large corpus of english text.

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

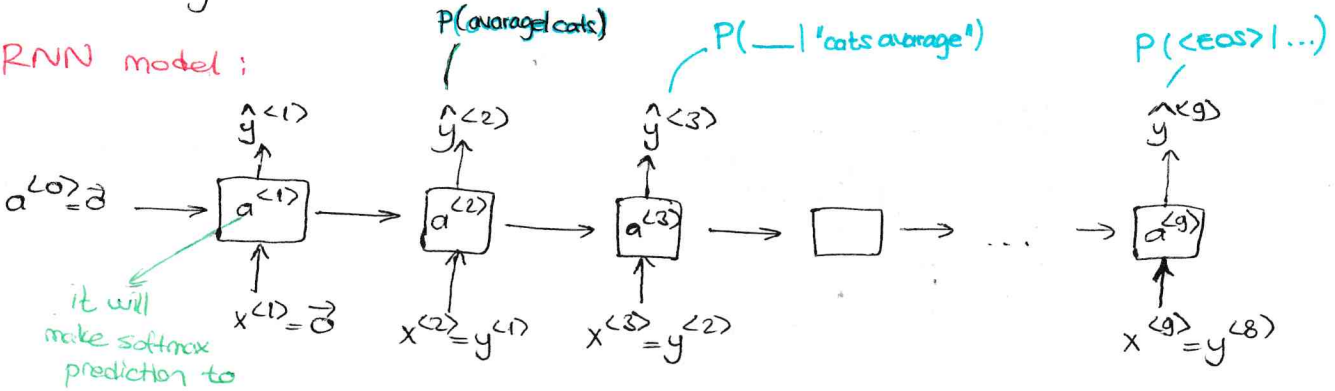
$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ .... $y^{<8>}$ $y^{<9>}$

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

<UNK>

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

RNN model :



it will make softmax prediction to try to figure out what is the prob. of the first words y.

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

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

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

Sampling Novel Sequences

Word-level language model:

Vocabulary = [a, aaron, ..., zulu, <UNK>]

Character-level language model:

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

Pros:

- You don't ever have to worry about unknown word tokens.
- also used in more specialized applications where you have a more specialized vocabulary.

Cons:

- You end up with much longer sequences.
- Not as good as word level lang. models at capturing long range dependecies between how the earlier parts of the sentences also affect the later part of the sentence.
- more computationally expensive to train.
- They are not in widespreade use today.

Vanishing Gradients with RNNs

The (cat), which already ate ....., (was) full.
The (cats), which already ate ....., (were) full.

↓

This is right or wrong, it's just very difficult for the area to backpropagate all the way to the beginning of the sequences and therefore to modify how the neural network is doing computations earlier in the sequence.
This is weakness of RNN algorithm.

When exploding gradients happened, you see NaN's. Use gradient clipping.

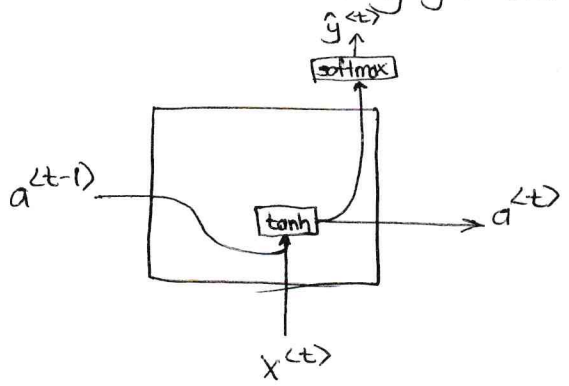
↓

looksat gradient vectors, and if it is bigger than some threshold, rescale some of your gradient vector so that it is not too big.

robust solution

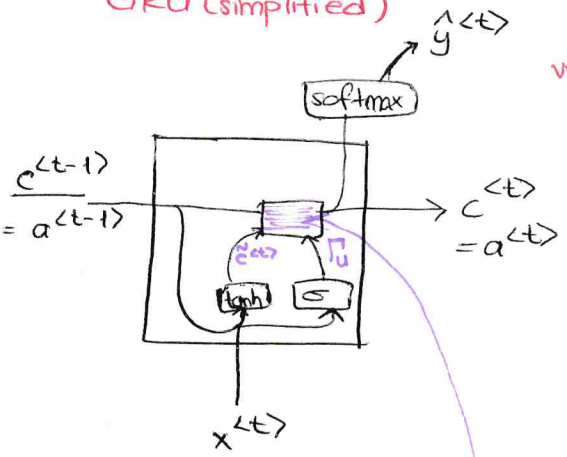
Gated Recurrent Unit (GRU)

It's a solution for vanishing gradient problems.



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

GRU (simplified)



vector

c = memory cell

$c^{<t>} = a^{<t>}$ → output activation value

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

↓ candidate of replacing c of t

$\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$ → tells you of this dimensional memory cell which are the bits you want to update

↓ update gate $(0,1)$

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

element-wise multiplication

Full GRU

tells how relevant is $c^{<t-1>}$ to $c^{<t>}$

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

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

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

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

↓

in literature

Long Short Term Memory (LSTM)

It is more powerful and more general version of GRU

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

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

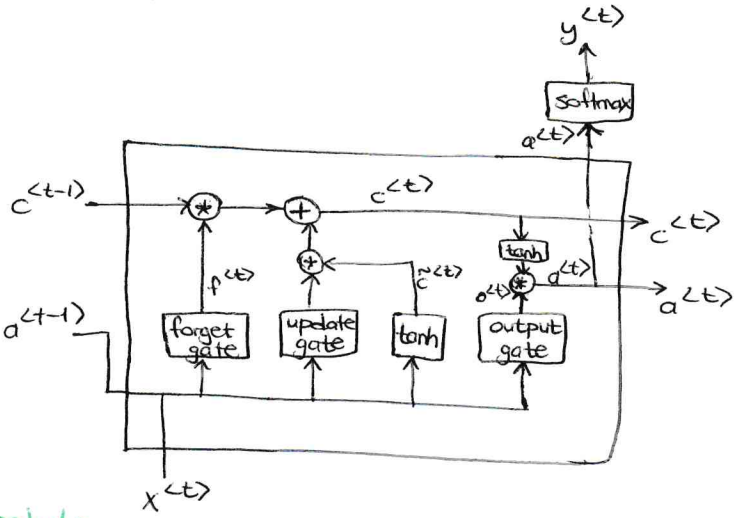
$$\Gamma_f = \sigma\left(W_f\left[a^{\langle t-1 \rangle}, x^{\langle t \rangle}\right] + b_f\right)$$

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

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

$$a^{\langle t \rangle} = \Gamma_o * \tanh c^{\langle t \rangle}$$

$c^{\langle t-1 \rangle}$ (peephole connection)

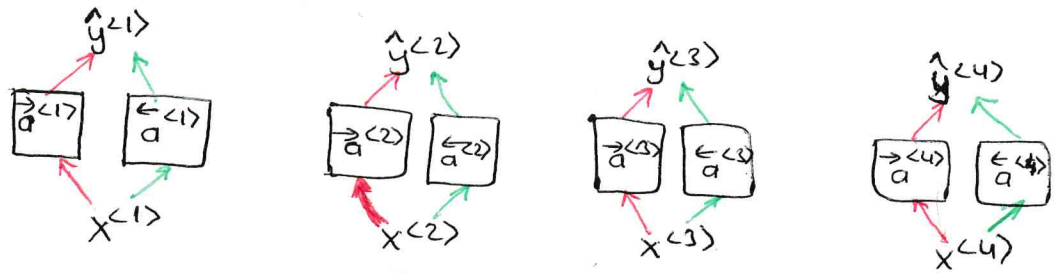


- LSTM came much earlier. GRUs were relatively recent invention
- GRU is simpler model, it is actually easier to build or scale a much bigger network,
- GRU has 2 gates, so computationally, it runs a bit faster.
- LSTM is more powerful and more effective since it has 3 gates instead of 2.

Andrew NG thinks LSTM has been the historically more proven choice.

Bidirectional RNN

also can be GRU LSTM



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

Disadvantage: You need the entire sequence of data before you can make predictions anywhere.

Introduction to Word Embeddings

COURSE 5
WEEK 2

Word Representation

$V = [a, aaron, \ldots, zulu, <UNK>]$ $\quad |V| = 10000$

1-hot representation

Man
(5391)

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

O_{5391}

One hot

Feature representation : word embedding

	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						
verb						
:						

300

e_{5391} (300,1) $\quad e_{9853}$ (300,1)

⇒ This allows it to generalize better across different words.

Using Word Embeddings

COURSE 5
WEEK 2

Named Entity Recognition Example:

Sally Johnson is an orange farmer.
Robert Lin is an apple farmer.
a durian cultivator.

~~Style~~ Transfer Learning:
→ 1B words - 100B words Bidirectional RNN
→ 100k words

Transfer Learning and Word Embeddings

1. Learn 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)
   → 10000 (sparse vector) → 300 (dense vector)
3. Optional: Continue to finetune the word embeddings with new data.

In practice, you would do this only if task 2 has pretty big data set.
If your label data set for step 2 is quite small, then usually, I would not bother to continue to finetune the word embeddings.

Word embeddings has been useful for:
- named entity recognition
- text summarization
- co-reference resolution
- parsing

less useful for:
- language modeling
- machine translation

Properties of Word Embeddings

	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_{5391} e_{man} e_{woman}

$e_{man} - e_{woman} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

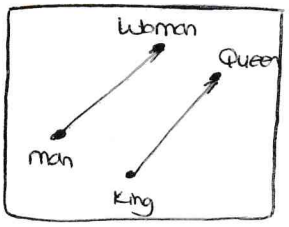
$e_{king} - e_{queen} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

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

Man → Woman vs King → Queen

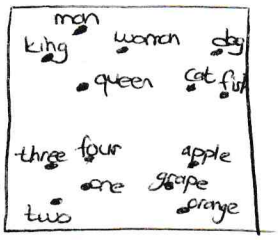
Analogies Using Word Vectors

COURSE 5
WEEK 2
3



300D

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



t-SNE → non-linear mapping

(It takes 300-D data and maps it in a very non-linear way to a 2D space.

Find word w : $\arg\max \; sim(e_w, e_{king} - e_{man} + e_{woman})$

Most commonly used similarity function is called COSINE similarity

Cosine Similarity

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

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

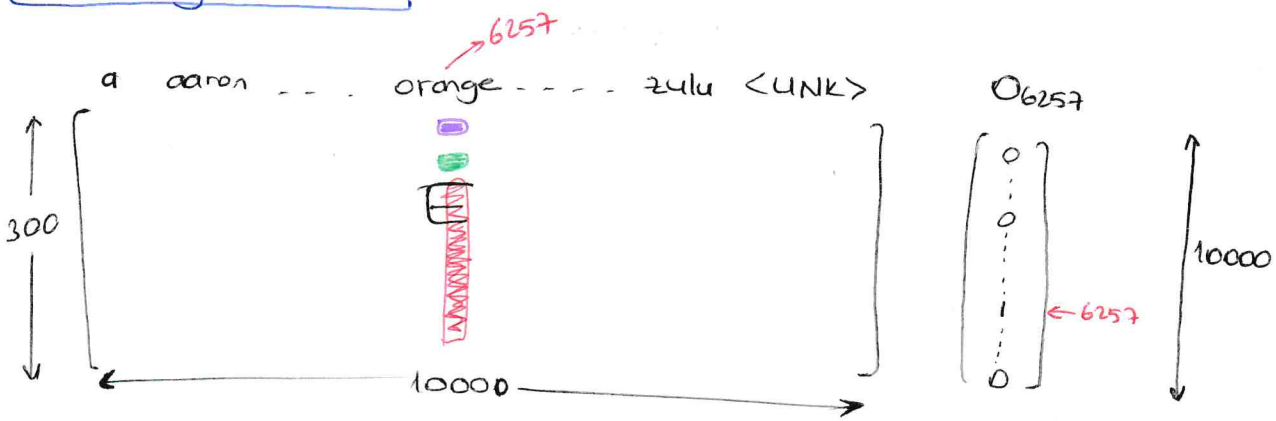
→ if u and v are very similar, their inner product will tend to be large.

Square Distance (Euclidian Distance) :

$$\|u - v\|^2$$

Main difference between these is how it normalizes the lengths of the vectors u and v.

Embedding Matrix



$E \cdot O_{6257} = e_{6257}$ (300,10k) (10k,1) → (300,1)

$E \cdot O_j = e_j$ = embedding for word j

! In practise, use specialized function to look up an embedding.

Learning Word Embeddings : Word2vec & GloVe

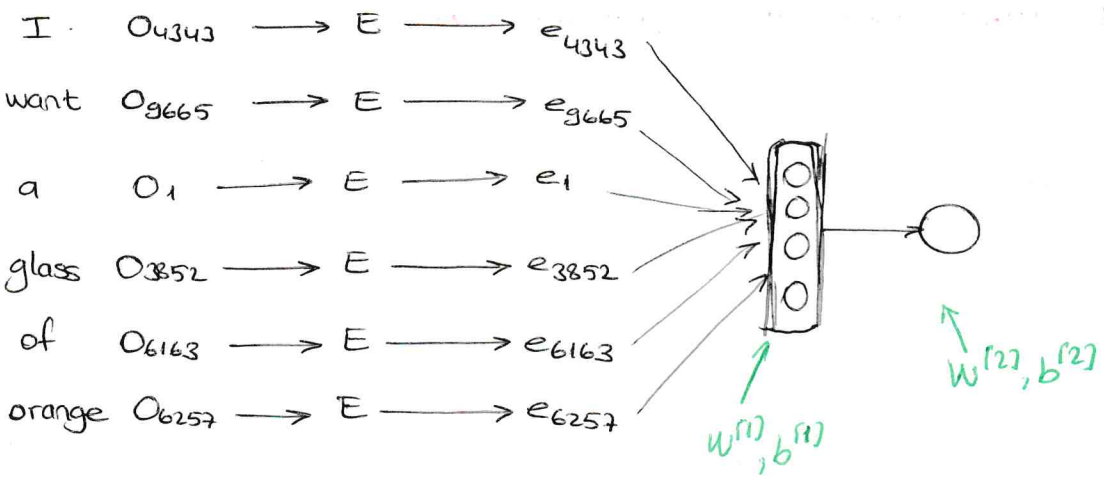
Learning Word Embeddings

COURSE 5
WEEK 2

Neural Language Model :

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

I want a glass of orange ___
4343 9665 1 3852 6163 6257



if we use 6 words, dimensions will be 1800 with their embeddings.

If 4 words are used, 1200 dimensions will be.

Other context/target pairs:

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

Context: Last 4 words.
4 words on left & right — a glass of orange ? to go along with
Last 1 word — orange ?
Nearby 1 word → idea of a Skip-Gram model — glass ?

Word2Vec

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

Skip-grams

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

	Context	Target	→ (within +10 word window)
randomly pick a word ←	orange	juice	⇒ supervised learning
	orange	glass	
	orange	my	

Model

COURSE 5
WEEK 2

5

Vocab size = 10 000k

$x \longrightarrow y$

Context c ("orange") $\longrightarrow$ Target t ("juice")

6257 4834

$O_c \longrightarrow E \longrightarrow e_c \longrightarrow O \longrightarrow \hat{y}$

$e_c = E \cdot O_c$ softmax

Softmax : $p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}}$

θ_t = parameters associated with a output t

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

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

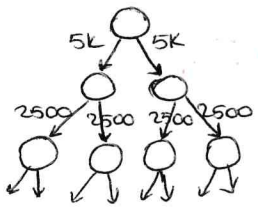
Problems with softmax Classification

$p(t|c) = \frac{e^{\theta_t^T \cdot e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}}$

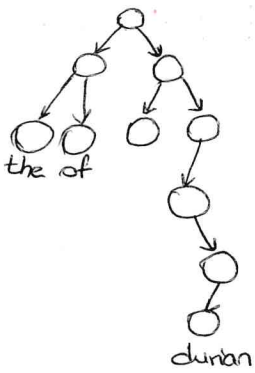
The primary problem is computational speed.

Solution : Hierarchical Softmax Classifier

computational size $\propto \log|V|$



In practice :



How to sample the context c?

→ the , of , a , and, to, ...

orange , apple , durian

P(c)

↙

In practice, the distribution of words p(c) isn't taken just entirely random for the training set purpose, instead , there are different heuristics in order to balance out something from the common words together with less common words.

Negative Sampling

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

Defining a new Learning problem :

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

context	word	target
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0

c (context) | t (possible target) | y (context target pair)

k = 5-20 smaller datasets

k = 2-5 larger datasets

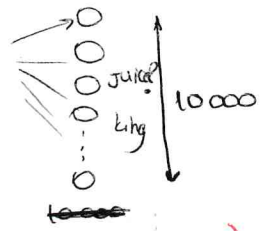
$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}}$$ → 10000 way softmax

parameter vector θ for each possible target word

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

→ sepeate parameter vector embedding vector for each possible context word

$O_{6257} \rightarrow E \rightarrow e_{6257}$



We are going to train one responding to the actual target word and then k randomly chosen negative examples.

→ 10000 binary classification problem

Selecting Negative Examples:

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

GloVe Word

COURSE 5
WEEK 2

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

Various Sequence to Sequence Architectures

Basic Models

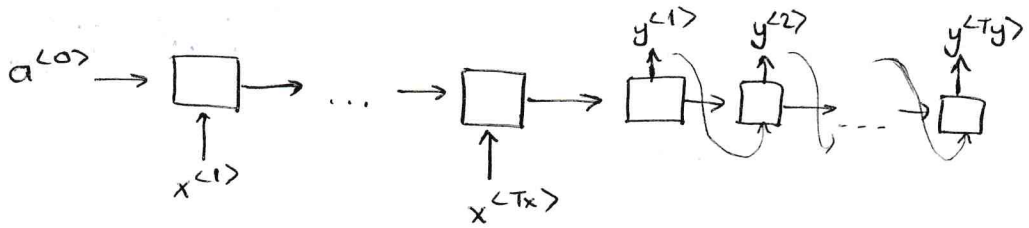
Sequence To Sequence model:

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

Jane visite I'Afrique en septembre.

→ Jane is visiting Africa in September.

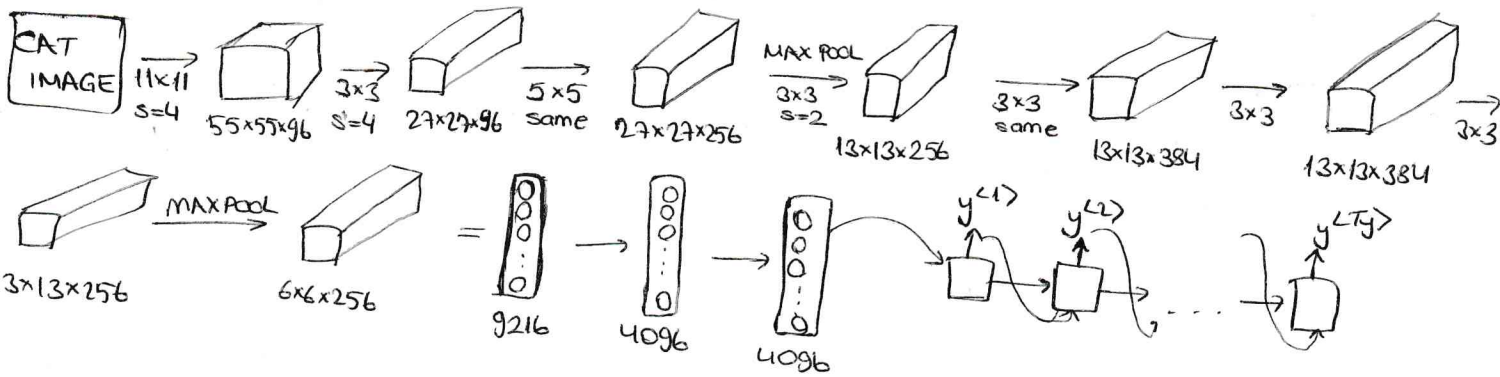
$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$



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

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

Image Captioning:



Picking The Most Likely Sentence

Finding the most likely translation

$P(y^{<1>}, \dots, y^{<Ty>} | x)$ (English, French)

Jane visite l'Afrique en septembre.

→ 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.

$$\underset{y^{<1>}, \dots, y^{<Ty>}}{\text{argmax}} \; P(y^{<1>}, \dots, y^{<Ty>} | x)$$

Greedy Search:

$P(\hat{y}^{<1>}, \hat{y}^{<2>}, \dots, \hat{y}^{<Ty>} | x)$

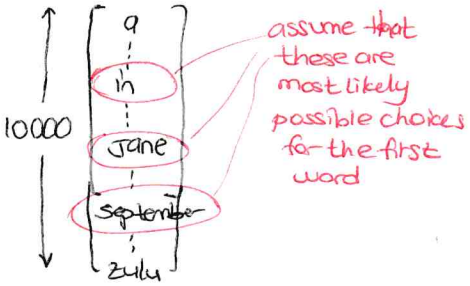
- $\hat{y}^{<1>}$: pick the best first word
- $\hat{y}^{<2>}$: After having the best first word, try to pick the best second word
- best third word

Beam Search

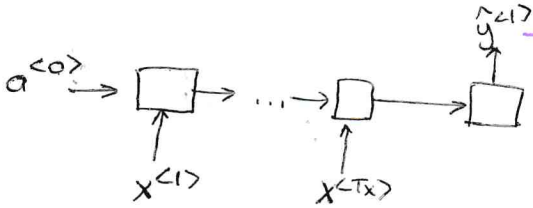
B = 3 (beam width)

↳ means beam search will consider 3 possibility at a time

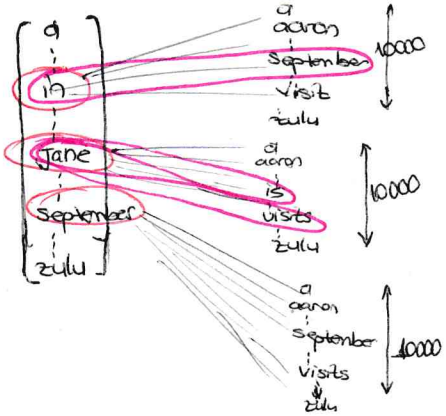
Step 1 :

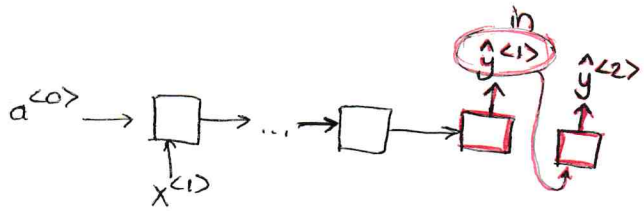


$P(y^{<1>}|x)$



Step 1 : Step 2 :



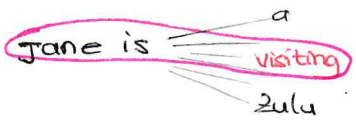


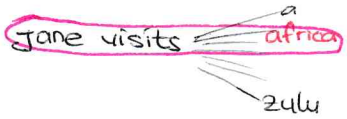
$P(y^{<1>}, y^{<2>}|x) = P(y^{<1>}|x) \cdot P(y^{<2>}|x, y^{<1>})$

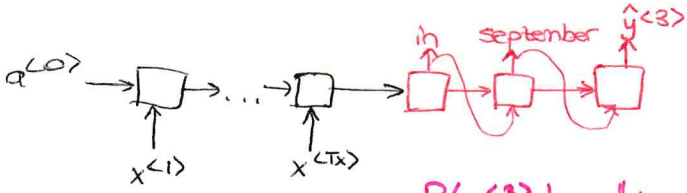
↳ evaluate 30000 options for the second word

Because of beam width equals 3, we instantiate 3 copies of the network to evaluate these partial sentence fragments and the output.

Step 3 :







$P(y^{<1>}, y^{<2>}|x)$

The outcome of this process hopefully will be :

Jane visits Africa in September. <EOS>

B = 1 ⇝ greedy search

B = 3 .. 5 .. 10 ⇝ beam search will find a much better output sentence than greedy search.

Refinements to Beam Search

Beam Search is a widely used algorithm in many production systems or in many commercial systems.

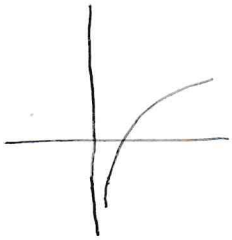
Length Normalization:

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

$P(y^{<1>} \dots y^{<T_y>} | x) = P(y^{<1>}|x) \cdot P(y^{<2>}|x, y^{<1>}) \dots P(y^{<T_y>}|x, y^{<1>} \dots y^{<T_y-1>})$

it will be very tiny result. So we take log of this products

$$\arg\max_y \sum_{y=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$



$\log P(y|x)$ should give the same result as maximizing $P(y)$ given x. $(P(y|x))$

If you have long sentence, the probability of that sentence is going to be low, because you're multiplying as many terms here.

$$\frac{1}{T_y} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

This significantly reduces the penalty for outputting longer translations

In practice:

$$\frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

$\alpha = 0.7$ ↓ hyperparameter

$\alpha = 1$ → completely normalizing by length

$\alpha = 0$ → not normalizing at all.

Beam Search Discussion:

Beam width B ? 10 100 1000 3000 (depends on the application)

The larger B is the more possibilities so you tend to get a better result.
the more computationally expensive your algorithm is. (Because of keeping all possibilities around)
It will be slow.
The memory requirements will grow, will also be computationally slower.

The small B is opposite of larger B.

Large B: better results, slower
Small B: worse result, faster

So for many applications, from beam width 1 to 3 to 10, there is huge gain as you go. But the gains as you go from 1000 to 3000 in beam width might not be as big.

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

Error Analysis in Beam Search

Example:

Jane visite l'Afrique en Septembre.

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

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

RNN → always tempting to collect more training data that never hurts

Beam Search → always tempting to increase beam width that never hurts

$$\text{RNN computes } P(y^*|x) \gtrless P(\hat{y}|x)$$

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

Beam search chose $\hat{y}$. But y^* attains higher $P(y|x)$.

Conclusion : Beam search is at fault.

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

y^* is a better translation than $\hat{y}$. But RNN predicted $P(y^*|x) < P(\hat{y}|x)$

Conclusion : RNN model is at fault.

The way you got $\hat{y}$ was you had RNN that was computing $P(y|x)$ and beam search's job was to try to find a value of $\arg\max_y P(y|x)$

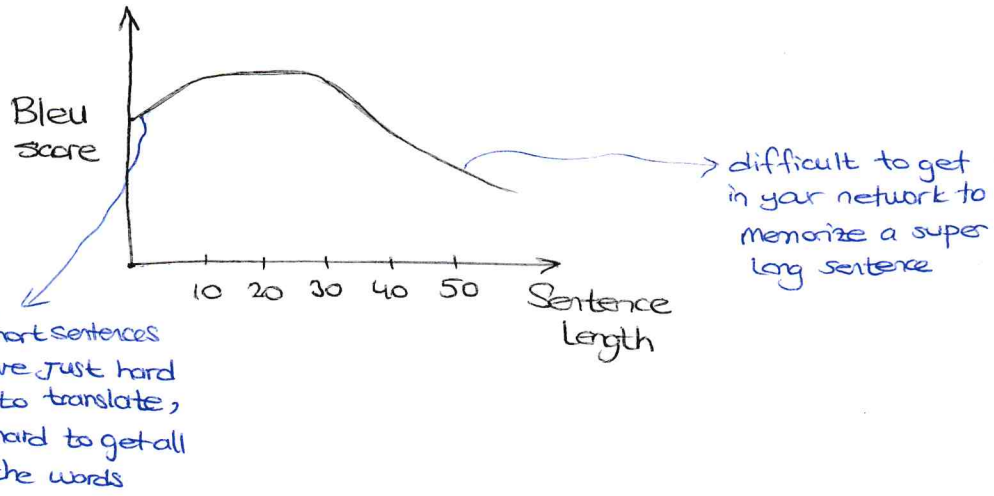
Error Analysis Process:

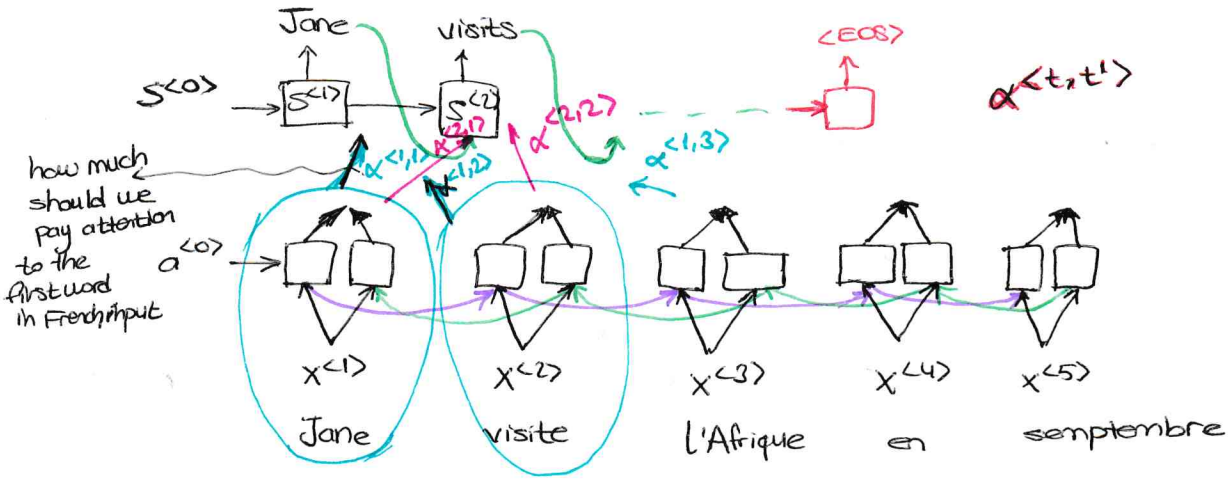
Human	Algorithm	$P(y^*\|x)$	$P(\hat{y}\|x)$	At fault
Jane visits Africa in September	Jane visited Africa last September	2×10^{-10}	1×10^{-10}	B
				R
				B
				R

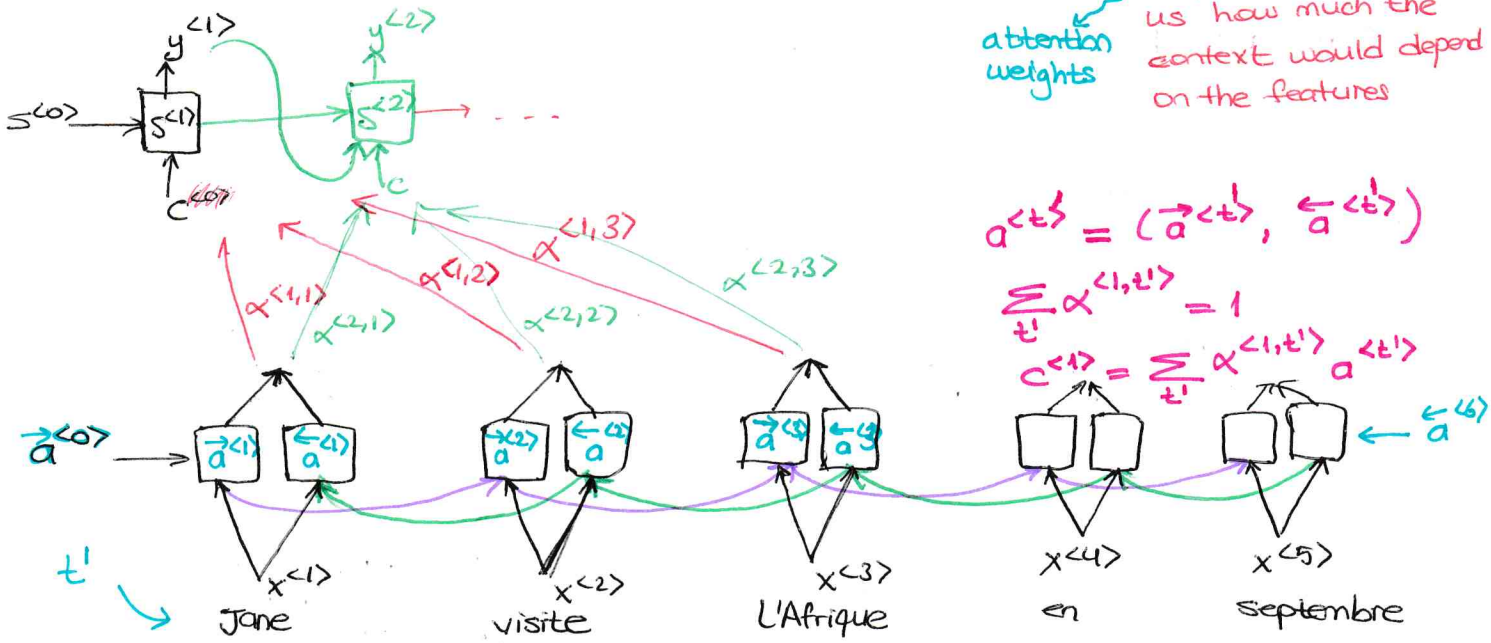
Figures out what fraction of errors are "due to" beam search vs RNN model.

~~The P~~ Attention Model

The Problem of Long Sequences :







$$a^{<t'>} = (\overrightarrow{a}^{<t'>}, \overleftarrow{a}^{<t'>})$$

$$\sum_{t'} \alpha^{<1,t'>} = 1$$

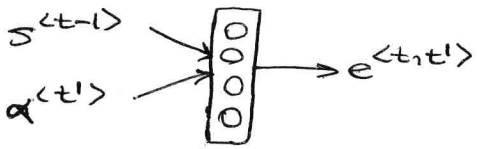
$$c^{<1>} = \sum_{t'} \alpha^{<1,t'>} a^{<t'>}$$

$\alpha^{<t,t'>}$ = amount of "attention" $y^{<t>}$ should pay to $a^{<t'>}$

Computing Attention $a^{<t,t'>}$:

$\alpha^{<t,t'>}$ = amount of attention $y^{<t>}$ should pay to $a^{<t'>}$

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$

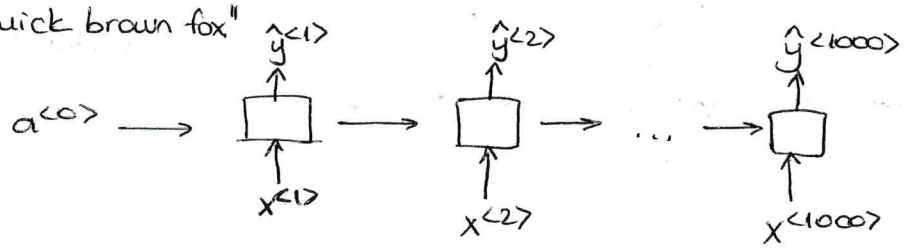


Speech Recognition

CTS cost for speech recognition:

(Connectionist temporal classification)

"the quick brown fox"



in practice, this network will usually be a bidirectional LSTM or bidirectional GRU and usually a deeper model.

ttt_h_eee___ ___qqq__

"space" "blank"

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

Production skills speech recognition system is a pretty significant effort and requires a very large dataset.

Trigger Word Detection