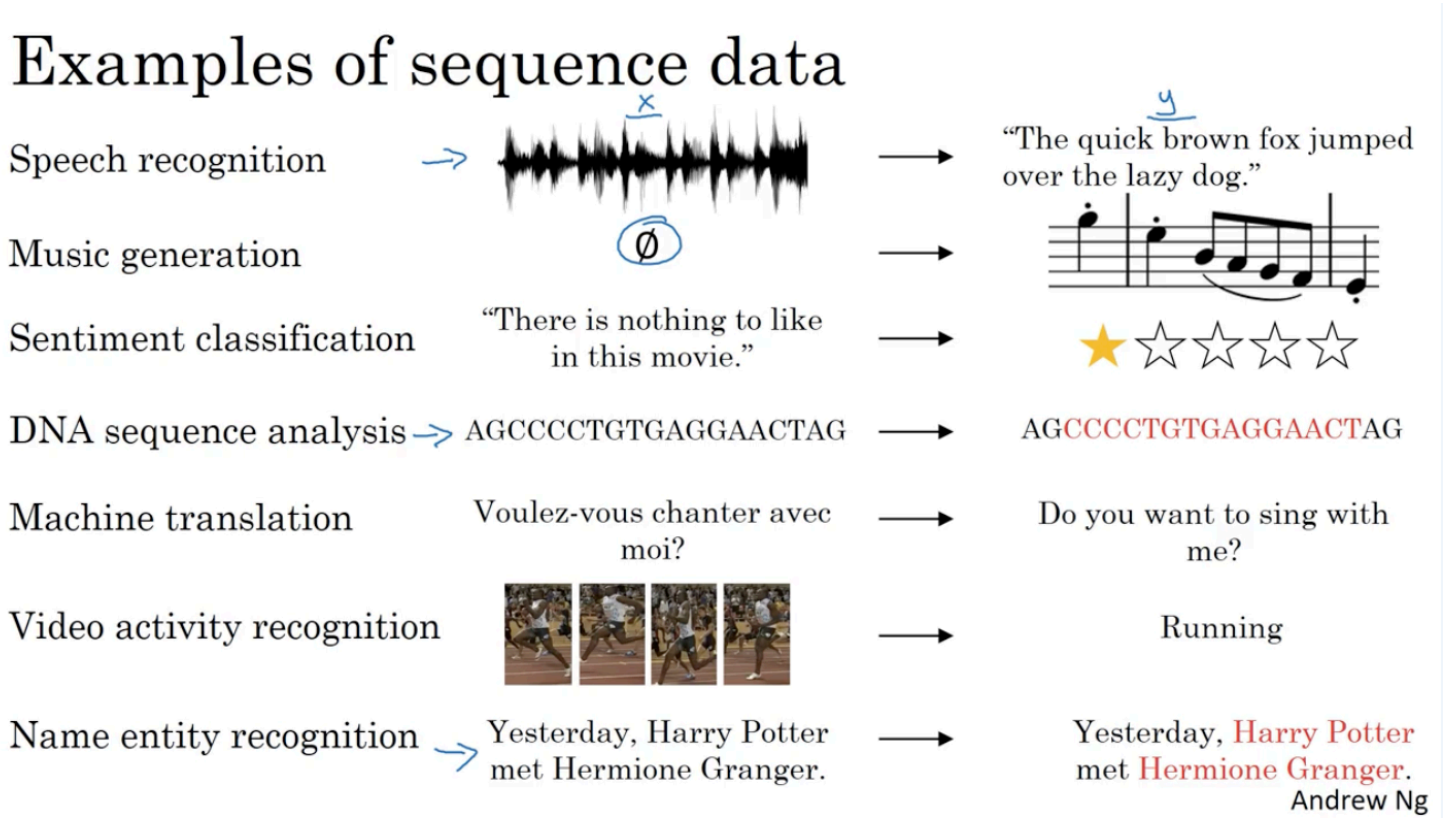
Sequence Models

Examples



in some cases only x/y is a sequence

Sequence Model Example: Named Entity Recognition (NER)

Input (x): "Harry Potter and Hermione Granger invented a new spell."
Output (y): 1 1 0 1 1 0 0 0 0 (Indicates presence/absence of a named entity)

Notation:

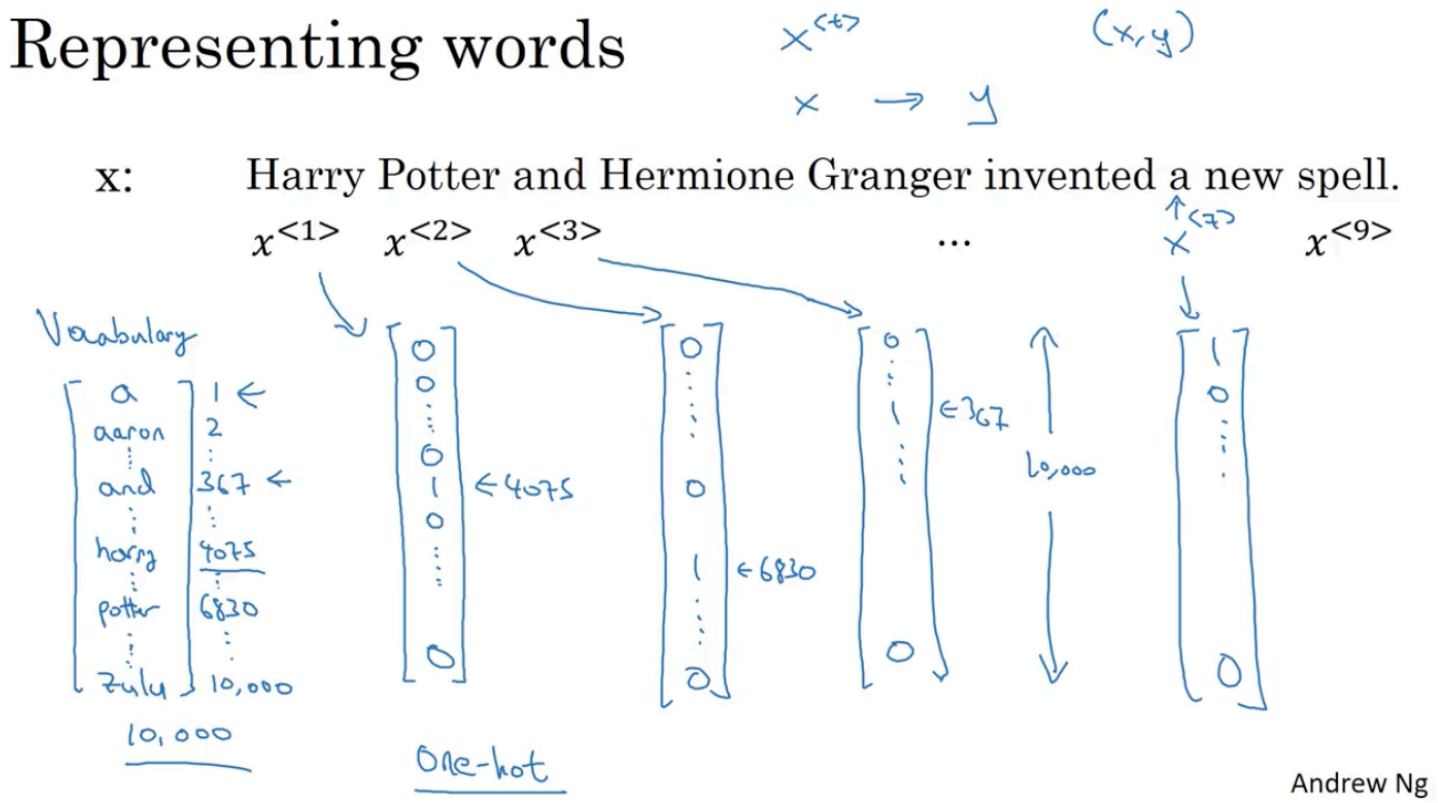
- $\mathbf{x^{<t>}}$**:** Word at position *t* in input.
- $\mathbf{y^{<t>}}$**:** Output label (0 or 1) at position *t*.
- $\mathbf{T_x}$**:** Input sequence length (here, 9).
- $\mathbf{T_y}$**:** Output sequence length (here, 9).
- $\mathbf{x^{(i)<t>}}$: Word at position *t* in the *i*-th input sequence.
- $\mathbf{y^{(i)<t>}}$: Output label at position *t* of the *i*-th sequence.
- $\mathbf{T_x^{(i)}}$: Length of the *i*-th input sequence.

Task: NER - Identify entities (names, dates, etc.) in text.

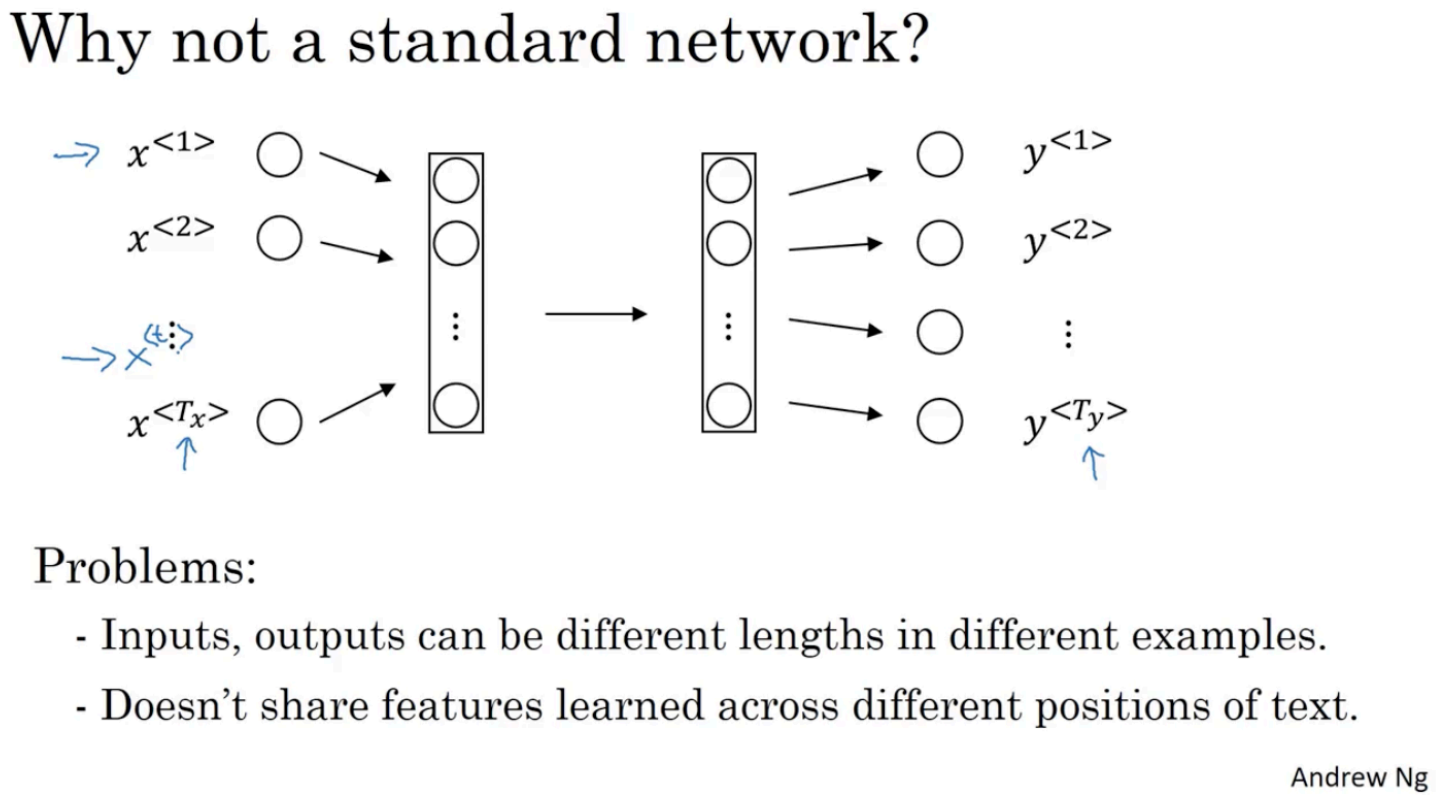
Note: Simple word-by-word classification is insufficient; context is crucial (e.g., "Harry Potter" vs. "Potter"). Better NER uses B- (beginning) and I- (inside) tags.

dictionary sizes: at least around (for commercial) 30k-50k, bigger uses/models vocabulary: millions.

Notation for Representing Words

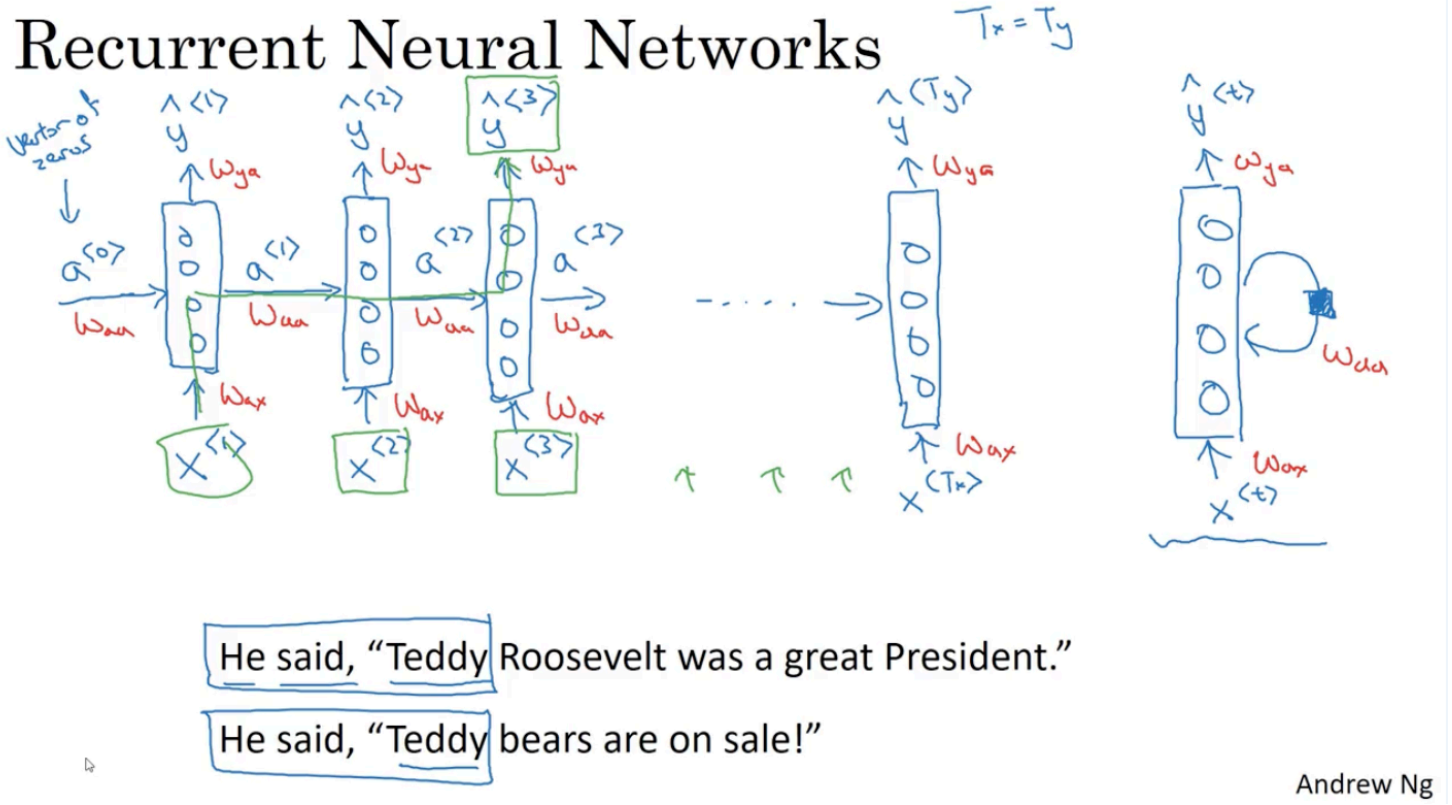


Recurrent Neural Network Model

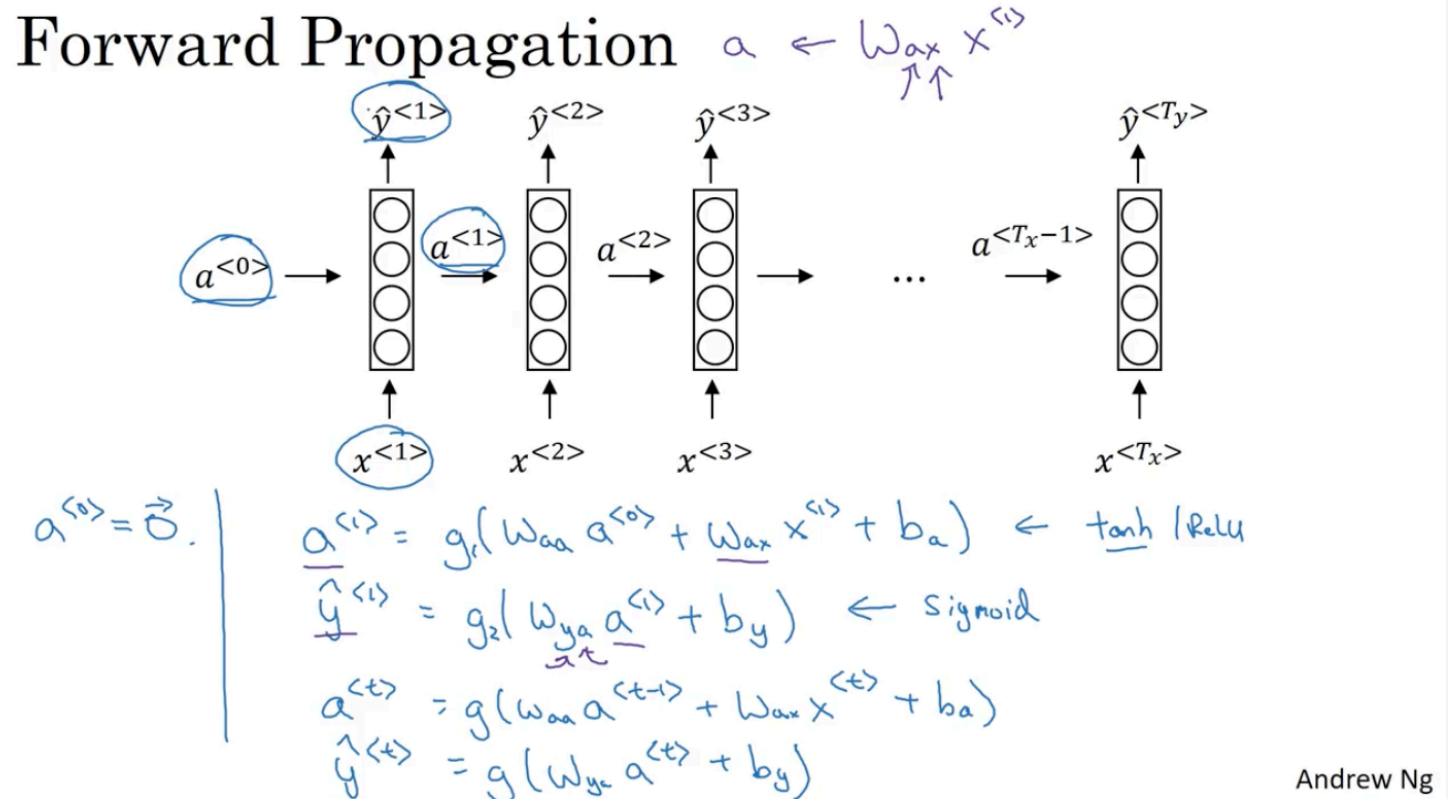


10k dimensional one-hot vector

- enormous amount of parameters

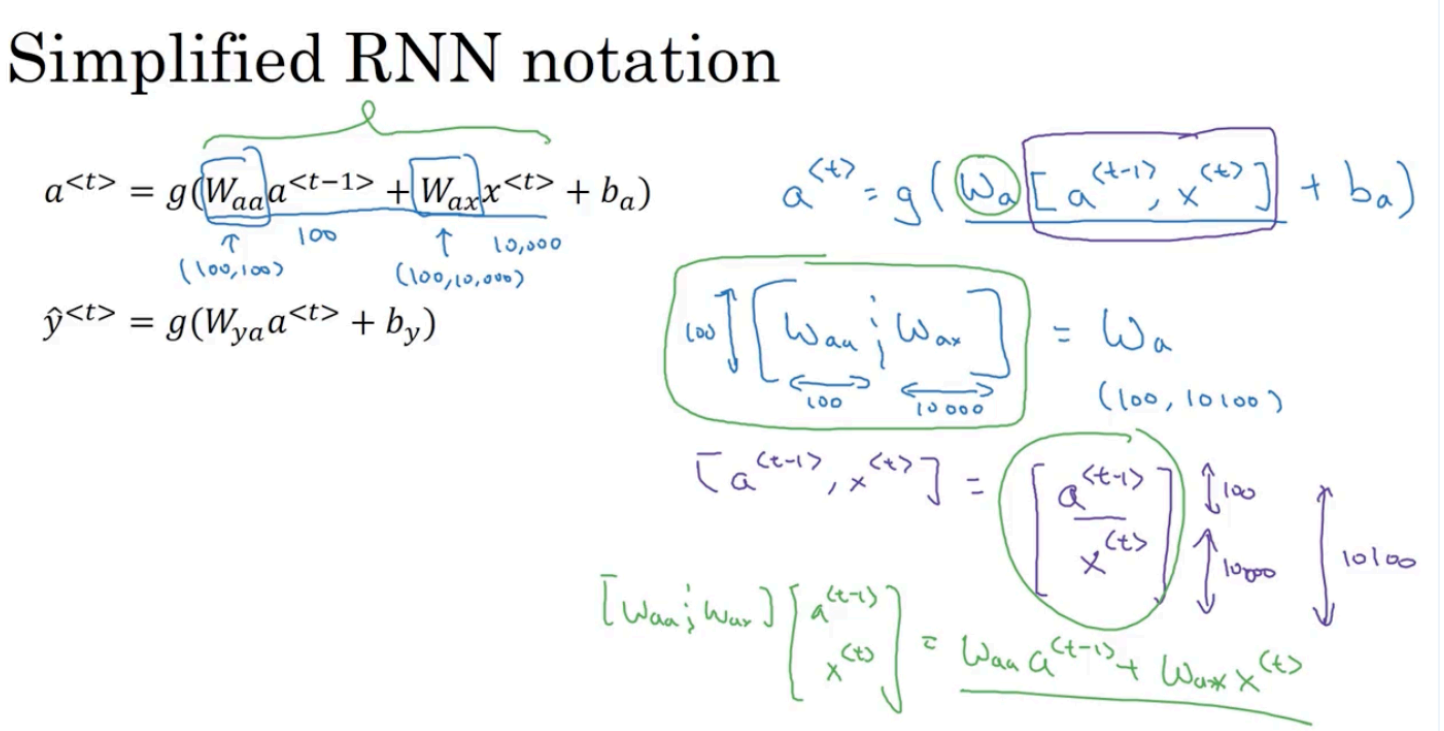


uni directional recurrent neural network

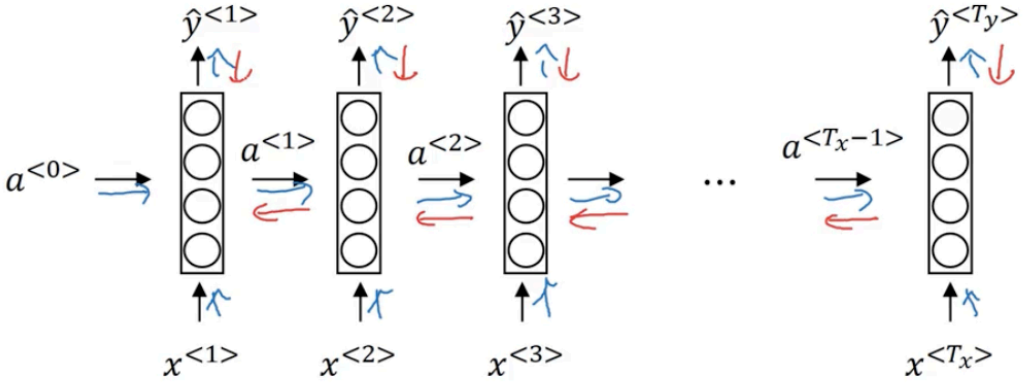


activation function, depends on what type of task - binary classification: sigmoid, though there are others for other tasks

Simplified RNN Notation

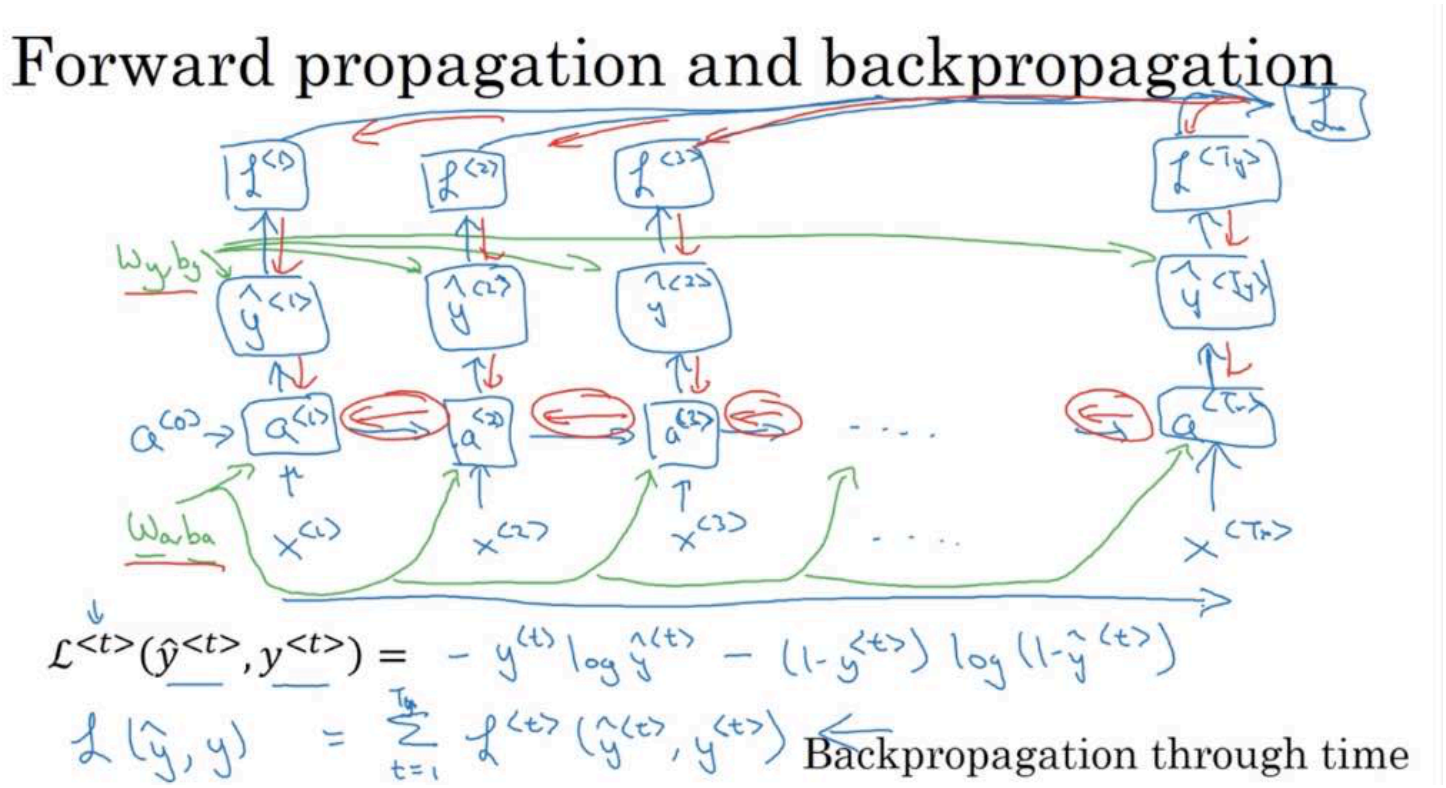


Backpropagation Through Time



basically the opposite direction of forward propagation

RNN: Forward and Backpropagation Through Time (BPTT)



Forward Propagation:

- Input sequence: $x^{<1>}, x^{<2>}, ..., x^{<T_x>}$
- Hidden states: $a^{<t>}$ (calculated sequentially, incorporating previous hidden state $a^{<t-1>}$ and current input $x^{<t>}$. *W_{aa}, W_{ax},b_a are involved, image is a simplified version*).
- Outputs: $\hat{y}^{<t>}$ (calculated from hidden state $a^{<t>}$. *W_{ya},b_y are involved, image is a simplified version*).
- Loss at each time step: $L^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>}\log(\hat{y}^{<t>}) - (1-y^{<t>})\log(1-\hat{y}^{<t>})$ (Cross-entropy loss)

Backpropagation Through Time (BPTT):

- Total Loss: $L(\hat{y}, y) = \Sigma_{t=1}^{T_y} L^{<t>}(\hat{y}^{<t>}, y^{<t>})$
- Gradients are calculated and propagated backward through time, from $t = T_y$ down to $t = 1$.
- The key characteristic is the backpropagation *through* the recurrent connections (the red arrows in the image, representing the flow of gradients through the hidden state dependencies across time).
- This unrolling represent the model computation dependencies.

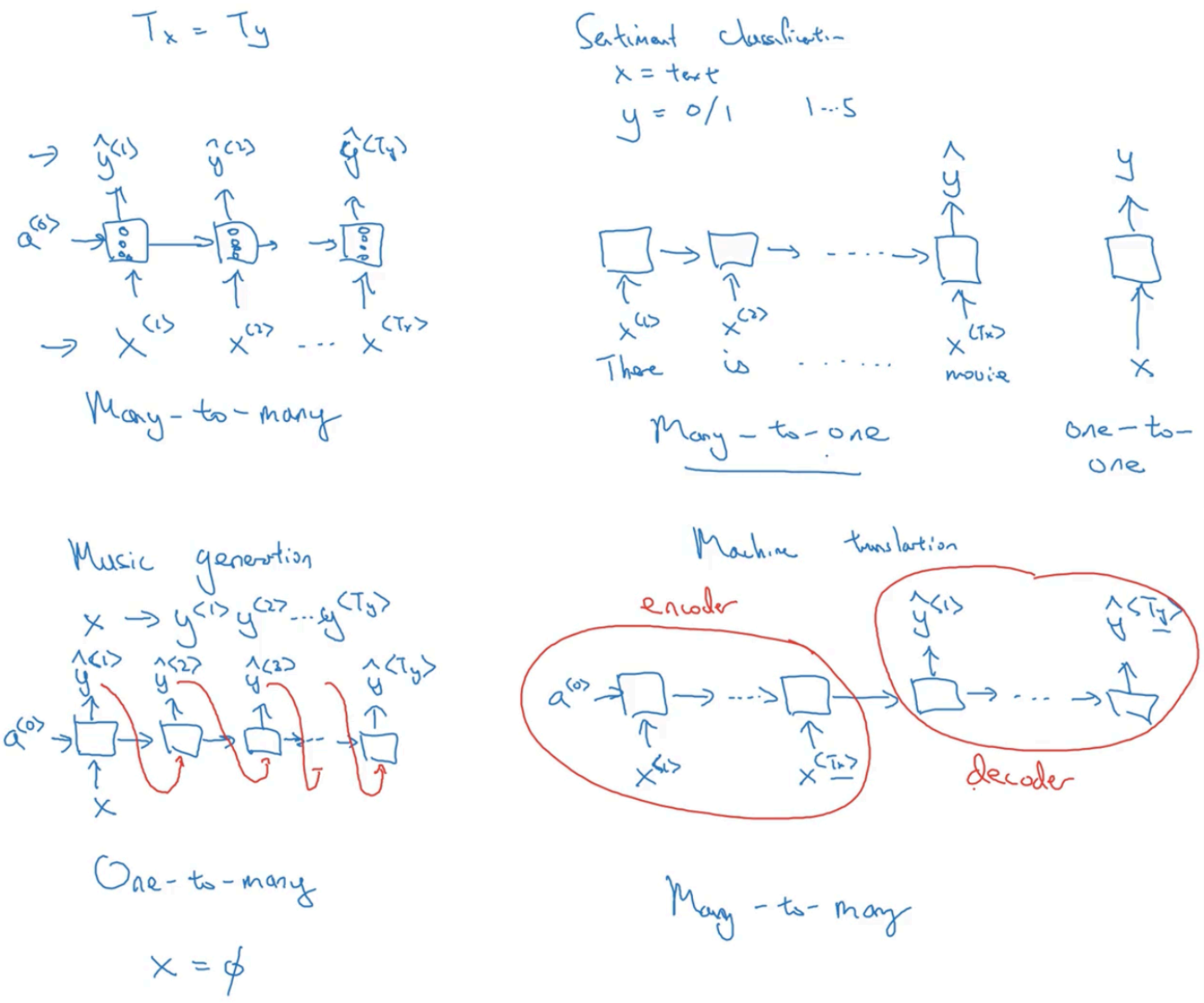
Note that the provided image represents an *unrolled* Recurrent Neural Network. W_{ax}, b_a, W_{aa}, W_{ya} and b_y parameters are omitted.

Different Types of RNNs

output could be n - 0 - or a fixed number like 5 (star rating from a sentence - sentiment)
input output numbers do not match (english to french)

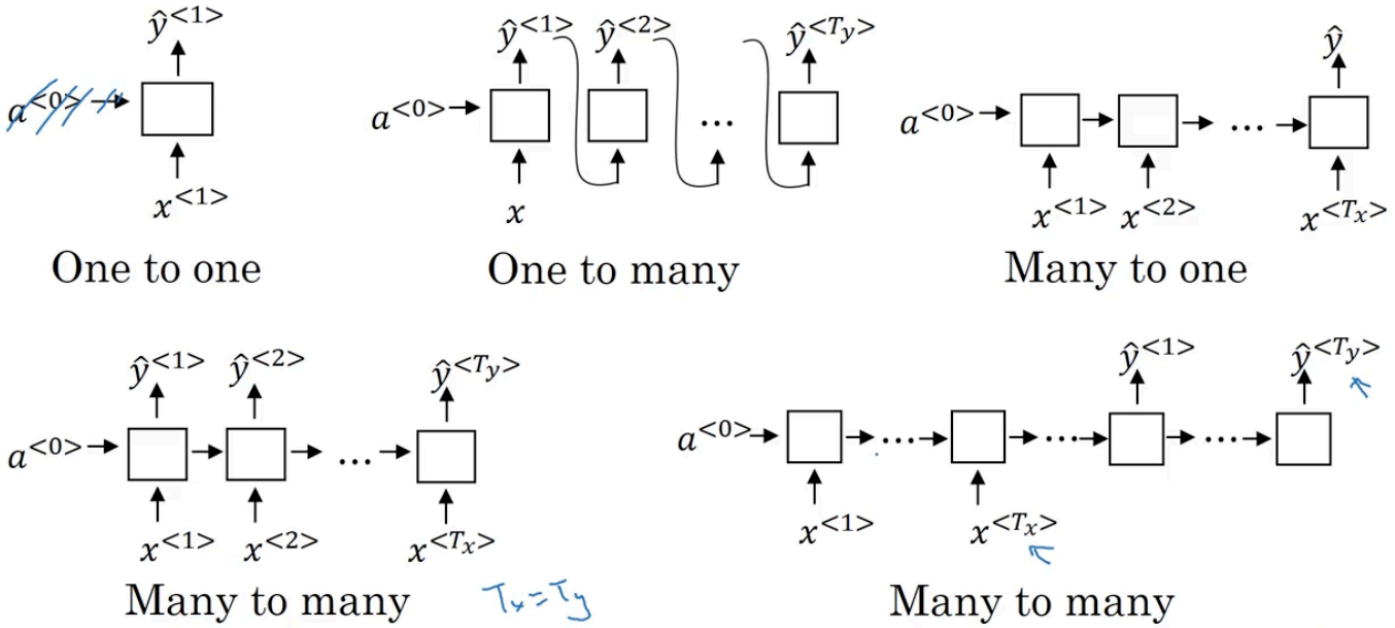
- we can modify the basic RNN architecture to address all these problems

Examples of RNN architectures



the famous encoder/decoder architecture

Summary of RNN types



Language Model and Sequence Generation

Language Modeling

Definition: A language model estimates the probability of a sequence of words, $P(y^{<1>}, y^{<2>}, ..., y^{<T_y>})$.

Application Example: Speech Recognition

- Given an audio input, a speech recognition system might generate multiple possible transcriptions.
- A language model helps choose the most likely (most probable) sequence.

Example:
Two possible transcriptions of the same audio:

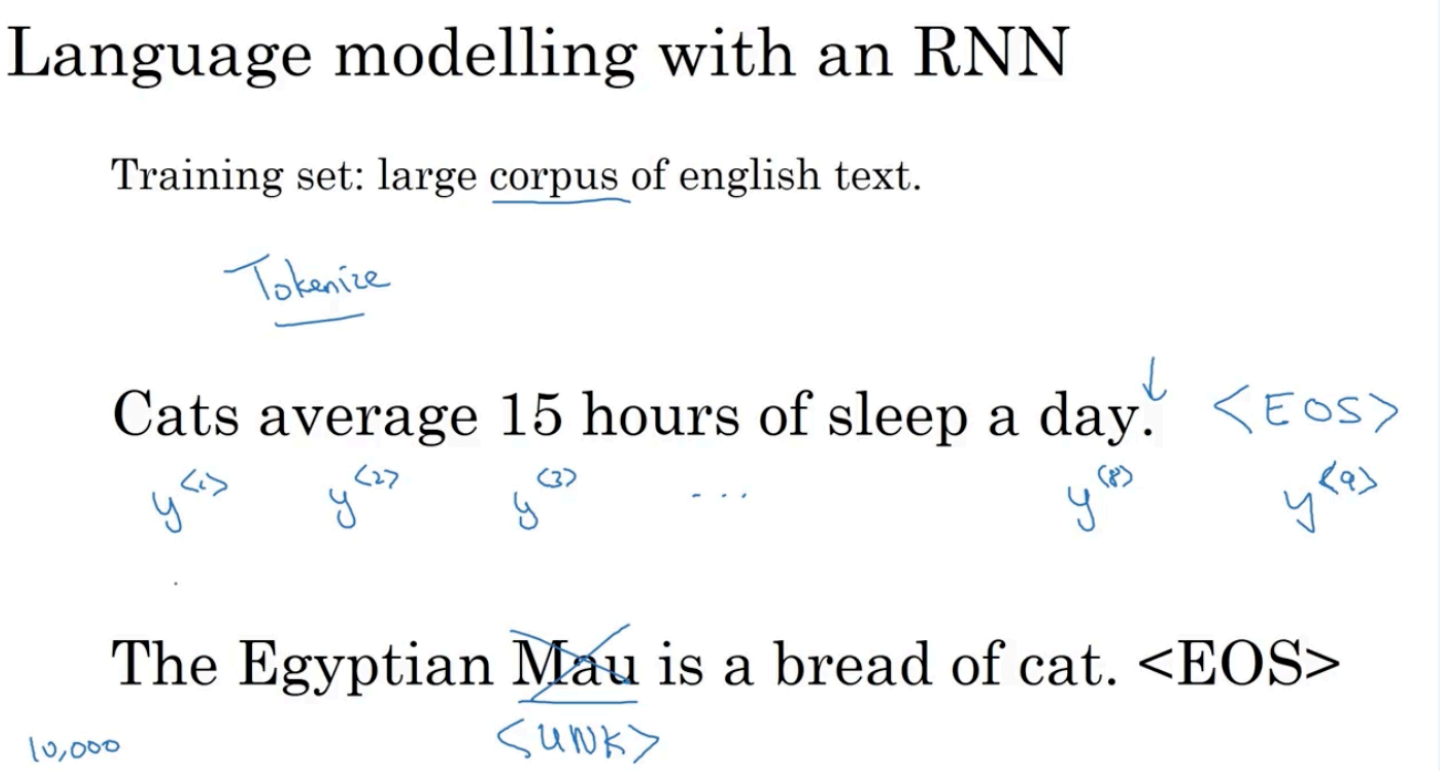
1. "The apple and pair salad."
2. "The apple and pear salad."

Language Model Probabilities:

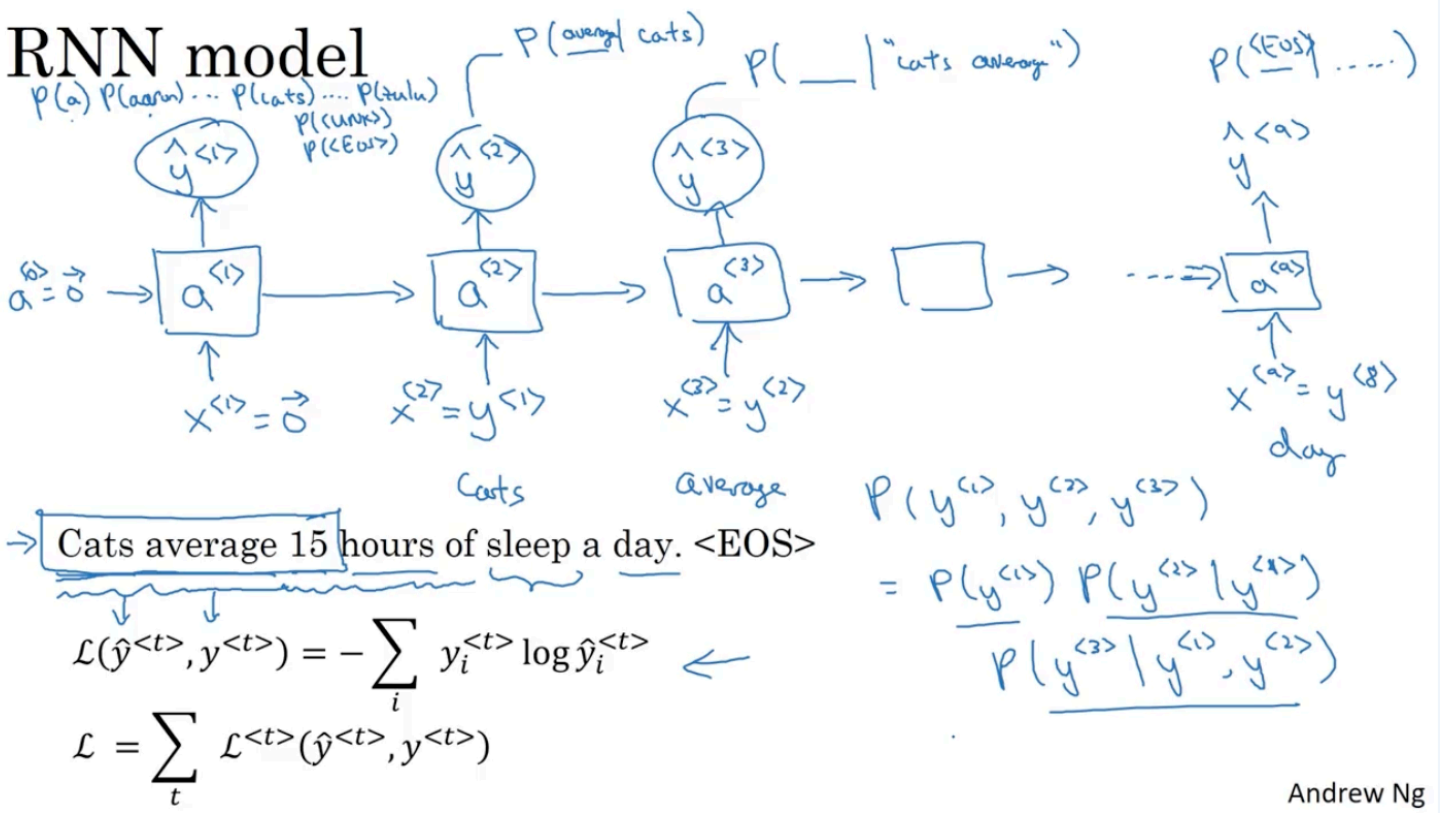
- P("The apple and pair salad") = 3.2×10^{-13}
- P("The apple and pear salad") = 5.7×10^{-10}

Conclusion: The language model assigns a higher probability to "The apple and pear salad," making it the more likely correct transcription. The model learned that "pear" as fruit is a likely word in this context, rather than "pair".

Tokenization

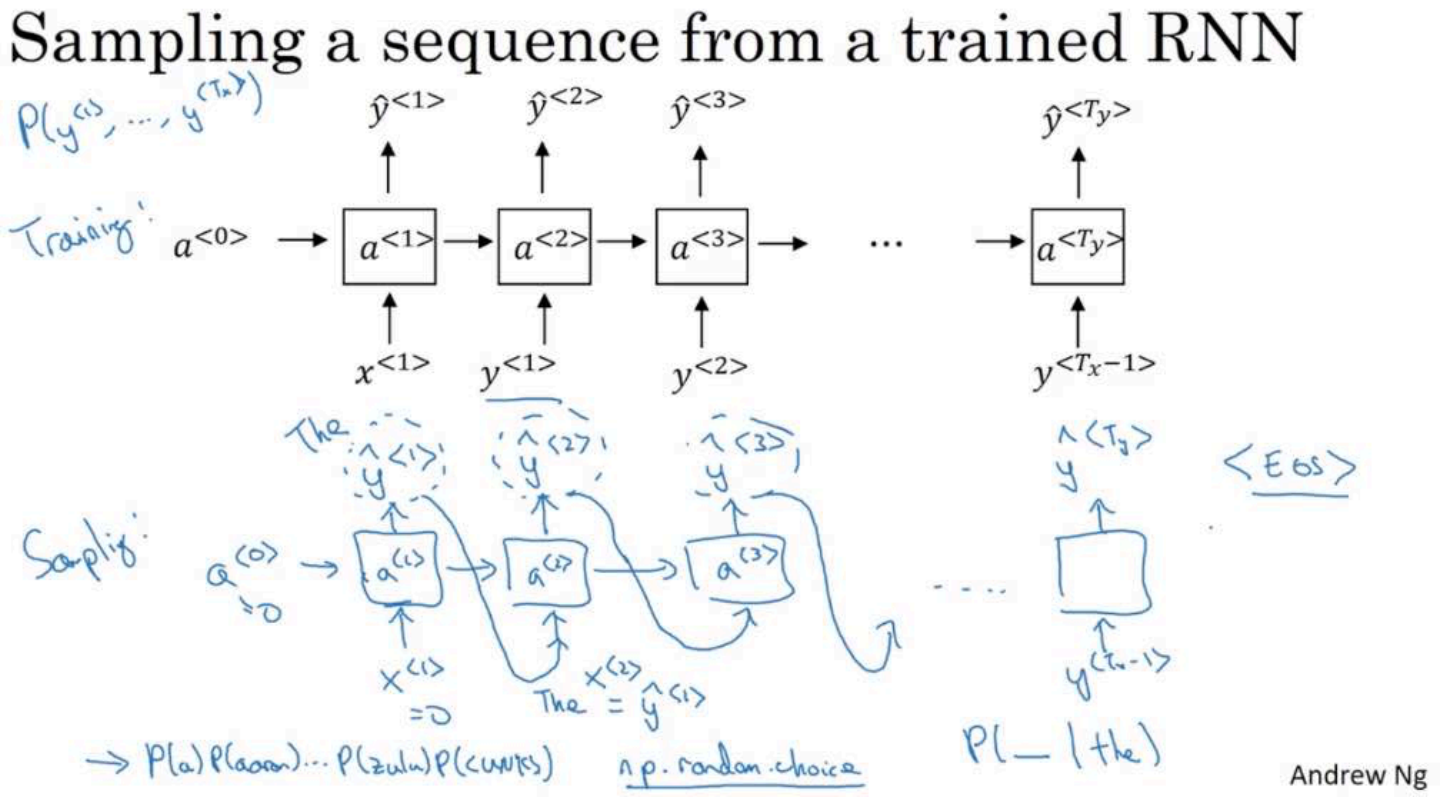


<'EOS> : end of sentence token - punctuation - <'UNK> unique token: names etc



softmax loss function
tries to predict next token.

Sampling Sequences from a Trained RNN Language Model



Process:

1. **Initialization:**
   - `a[0] = 0` (initial hidden state, often a zero vector).
   - `x[1] = 0` or `<SOS>` (start-of-sequence input).
2. **Iterative Generation (for each time step t):**
   - Input: Previous hidden state (`a[t-1]`) and previous *predicted* output (`y[t-1]`). *Note: During training, `y[t-1]` is the _correct* previous word; during sampling, it's the *sampled* word from the

previous step._
   - RNN computes: `P(y[t] | y[1], ..., y[t-1])` (probability distribution over vocabulary).
   - Sample a word from this distribution (e.g., using `np.random.choice`). This sampled word becomes `y[t]`.
   - `x[t+1] = y[t]` (The sampled word becomes the input for the next time step).
3. **Termination:** Stop when `<EOS>` is sampled or max length is reached.

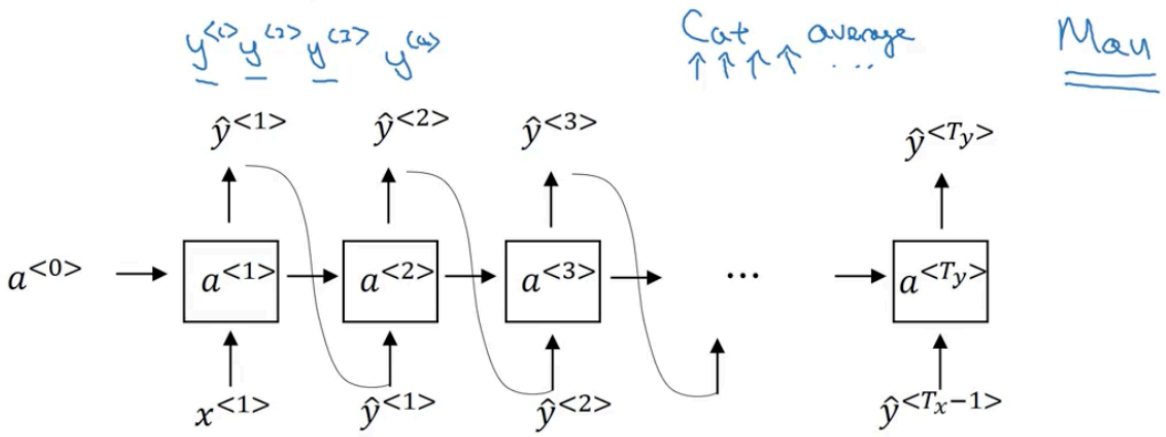
Key Difference: Training vs. Sampling

- **Training:** Uses *correct* previous word (`y[t-1]`) as input. Learns `P(y[t] | y[1], ..., y[t-1])`.
- **Sampling:** Uses *predicted* previous word (`y[t-1]` , initially sampled, then fed back) as input. Generates new sequences using the learned probability distribution.

Character-level language model

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

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



Andrew Ng

Sequence generation

News

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

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

"Concussion epidemic", to be examined. ←

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

Shakespeare

The mortal moon hath her eclipse in love.

And subject of this thou art another this fold.

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

For whose are ruse of mine eyes heaves.

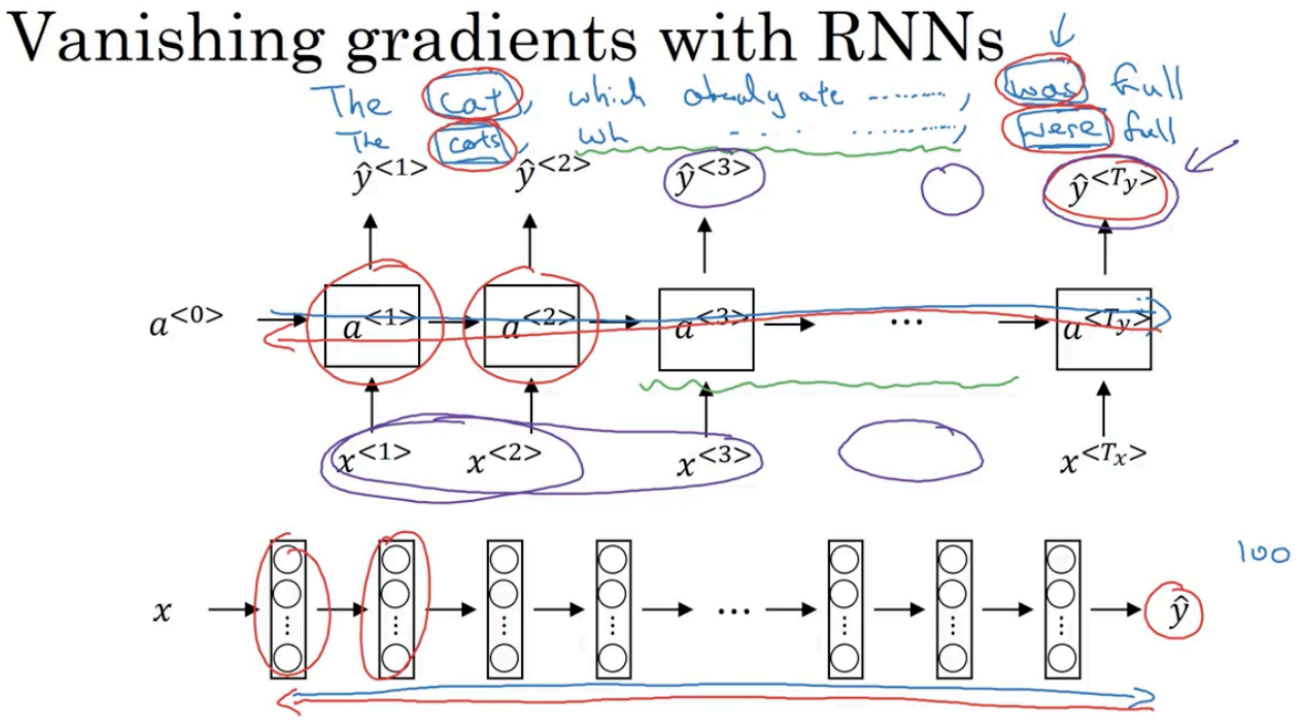
whatever is the input, is the output

Vanishing Gradients with RNNs

RNNs tend to have vanishing gradients problem

very early tokens, may have effect on later tokens, base RNN architecture is not good at preserving such effects to later 'layers'.

- the cat ..... was
- the cats ..... were



highly local, influenced mainly by inputs near that output
weakness of basic RNN

- if not addressed, not good at capturing long term dependencies.

also this can cause exploding gradients

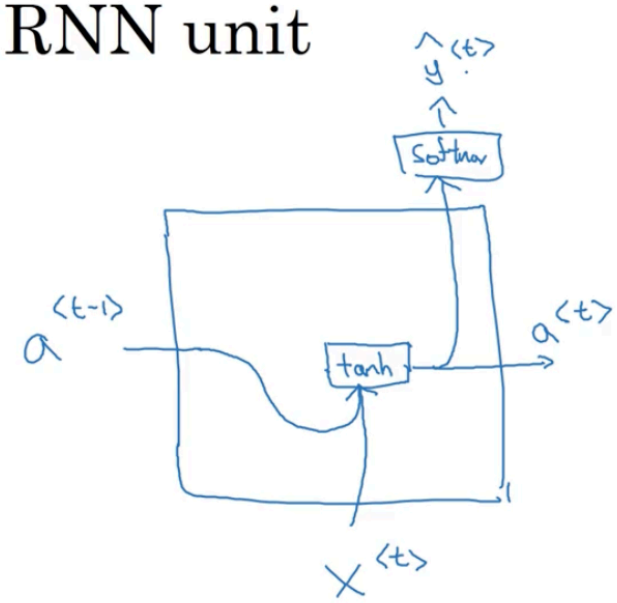
- easier to spot, parameters blow up - NaN
- soln: gradient clipping - look gradient vector, if bigger than X, clip. Robust solution

vanishing gradients are much harder to solve

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

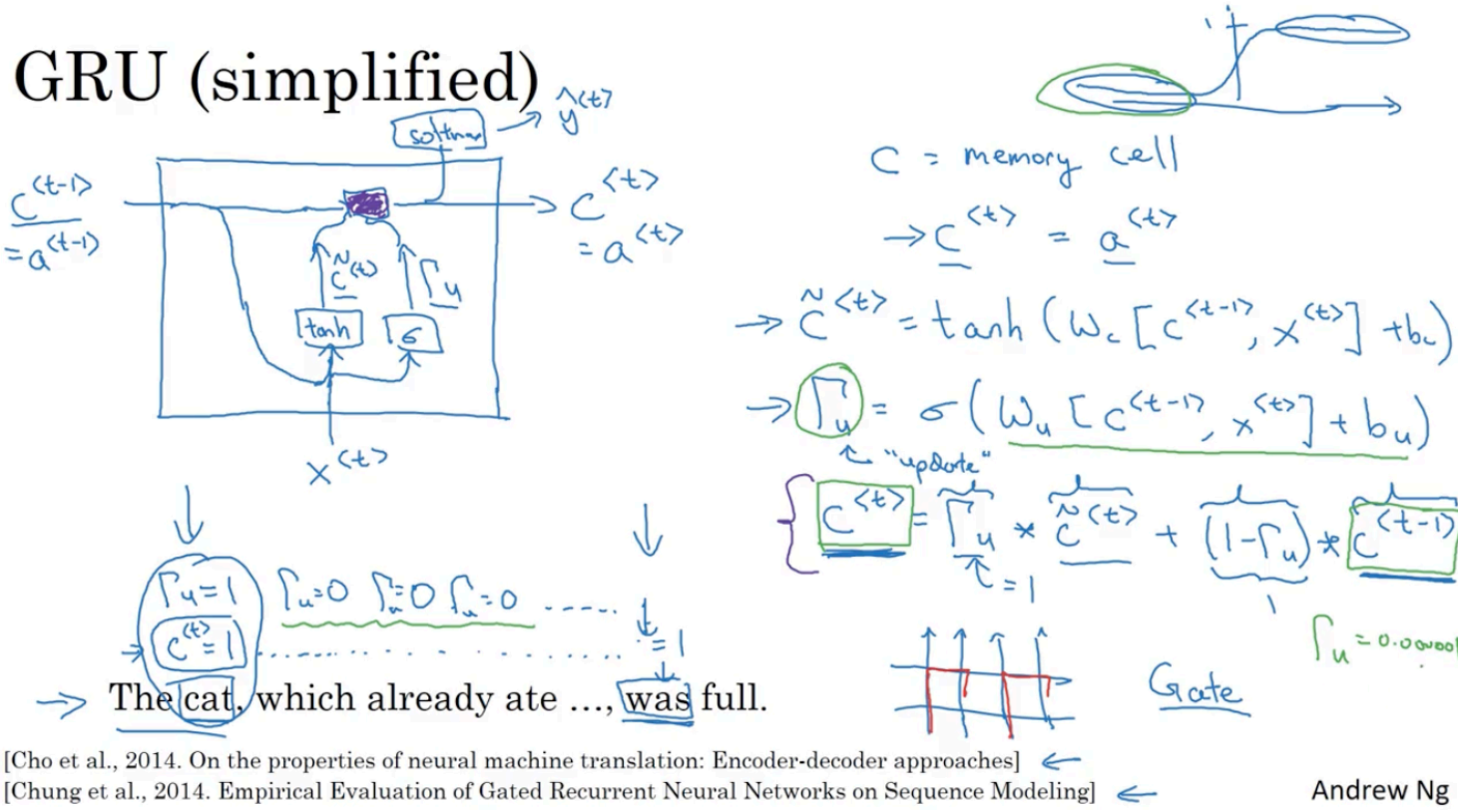
Gated Recurrent Unit (GRU)

RNN Unit:



GRU (simplified):

for LSTM, c of t and a of t are not equal, though that is for later



[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]

Andrew Ng

!!!

very good at maintaining the value of the cell - not changing the memory value

c t can be a x-dimensional-vector (generally

gamma won't be exactly 0 and 1 mostly

you can choose some bits of the vector to keep them constant and choose some to update - with adjusting what (?)

Full GRU

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

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

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

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

LSTM

The cat, which ate already, was full.

Andrew Ng

Long Short Term Memory (LSTM)

Even more powerful than the GRU Unit

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

instead of one update gate controlling, for LSTM there are two gamma's, update & forgot also new output gate

GRU and LSTM

GRU

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

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

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

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

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

LSTM

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

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

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

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

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

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

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

Andrew Ng

LSTM in pictures

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

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

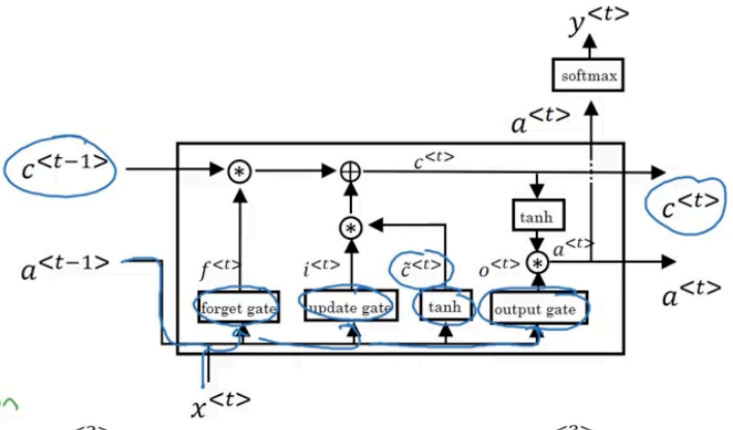
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

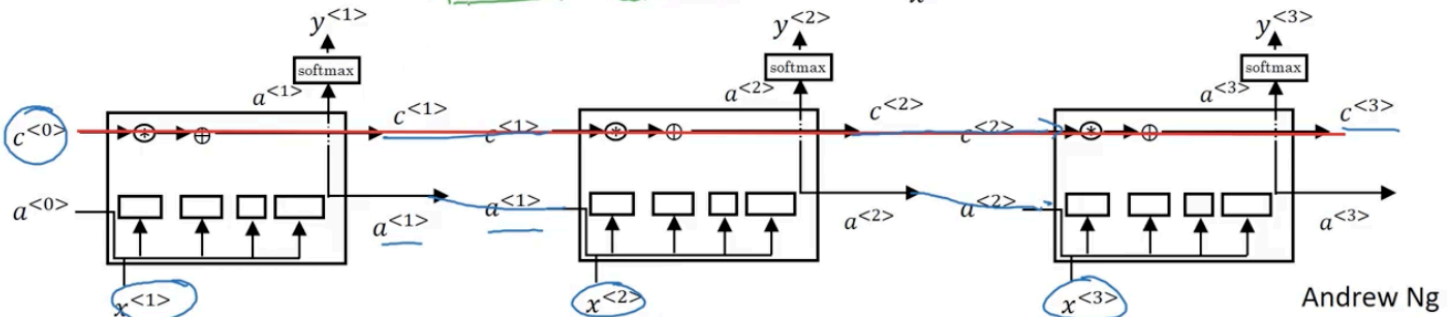
$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

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

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

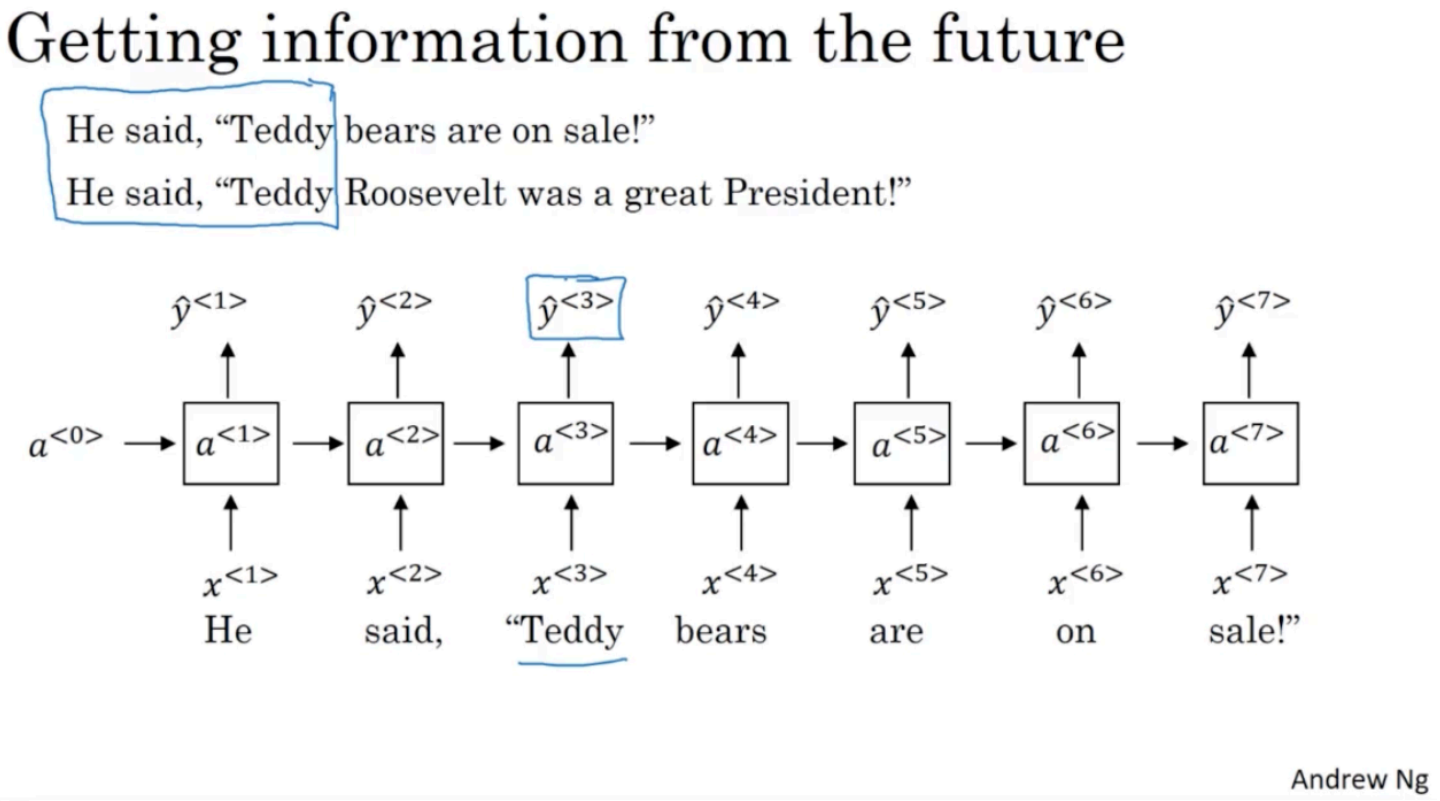
peephole connection



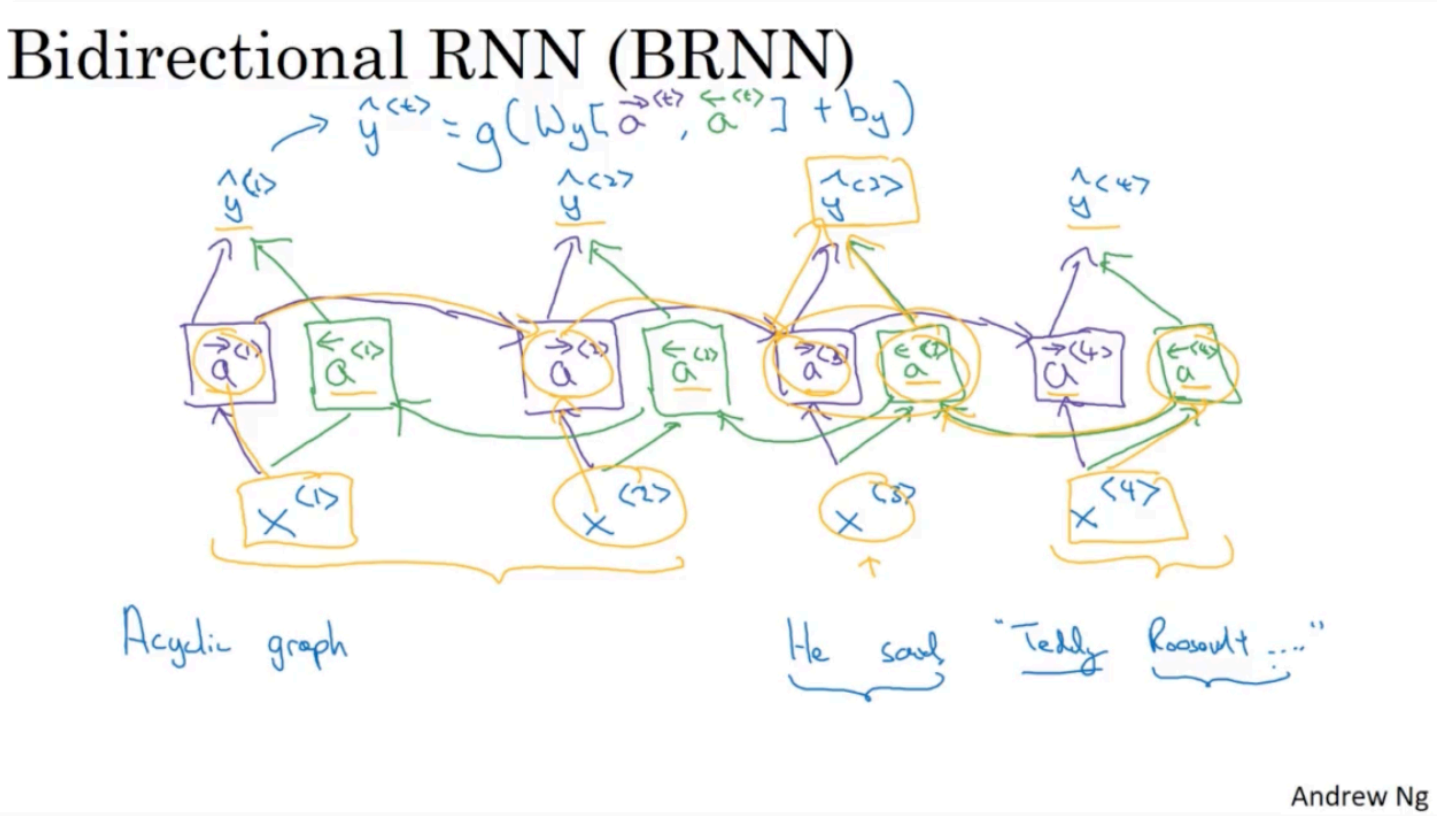


Andrew Ng

Bidirectional RNN



these blocks can be either basic RNN, GRU, or LSTM units.

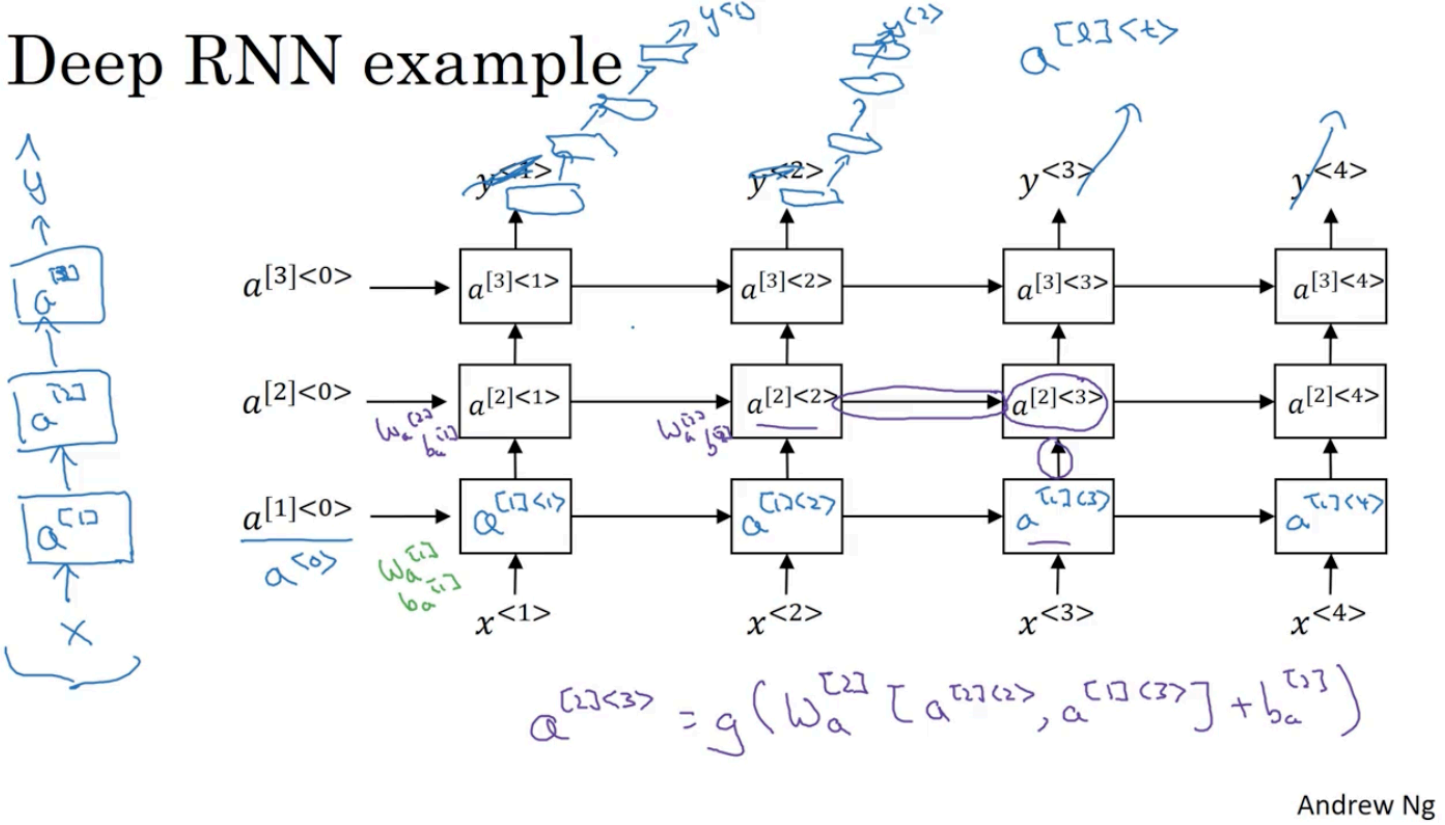


forward propagation, but it goes both to forward and 'backward'
information gets carried in both directions

- BRNN w\ LSTM blocks - used frequently for NLP problems
  Disadvantage: Need entire sequence of data to start processing / make prediction

Deep RNNs

RNNs work as they are already, deeper models make them better.



3 hidden layers
blocks does not have to be regular RNN blocks
there can be only horizontal connections - there are other versions of this network

---

What you should remember:

- The recurrent neural network, or RNN, is essentially the repeated use of a single cell.
- A basic RNN reads inputs one at a time, and remembers information through the hidden layer activations (hidden states) that are passed from one time step to the next.
  - The time step dimension determines how many times to re-use the RNN cell
- Each cell takes two inputs at each time step:
  - The hidden state from the previous cell
  - The current time step's input data
- Each cell has two outputs at each time step:
  - A hidden state
  - A prediction
- An LSTM is similar to an RNN in that they both use hidden states to pass along information, but an LSTM also uses a cell state, which is like a long-term memory, to help deal with the issue of vanishing gradients
- An LSTM cell consists of a cell state, or long-term memory, a hidden state, or short-term memory, along with 3 gates that constantly update the relevancy of its inputs:
  - A **forget** gate, which decides which input units should be remembered and passed along. It's a tensor with values between 0 and 1.
    - If a unit has a value close to 0, the LSTM will "forget" the stored state in the previous cell state.
    - If it has a value close to 1, the LSTM will mostly remember the corresponding value.

    - An **update** gate, again a tensor containing values between 0 and 1. It decides on what information to throw away, and what new information to add.
        - When a unit in the update gate is close to 1, the value of its candidate is passed on to the hidden state.
        - When a unit in the update gate is close to 0, it's prevented from being passed onto the hidden state.
    - And an **output** gate, which decides what gets sent as the output of the time step
- Very large, or "exploding" gradients updates can be so large that they "overshoot" the optimal values during back prop -- making training difficult
    - Clip gradients before updating the parameters to avoid exploding gradients
- Sampling is a technique you can use to pick the index of the next character according to a probability distribution.
    - To begin character-level sampling:
        - Input a "dummy" vector of zeros as a default input
        - Run one step of forward propagation to get $a\langle 1\rangle$ (your first character) and $\hat{y}\langle 1\rangle$ (probability distribution for the following character)
        - When sampling, avoid generating the same result each time given the starting letter (and make your names more interesting!) by using `np.random.choice`
- A sequence model can be used to generate musical values, which are then post-processed into midi music.
- You can use a fairly similar model for tasks ranging from generating dinosaur names to generating original music, with the only major difference being the input fed to the model.
- In Keras, sequence generation involves defining layers with shared weights, which are then repeated for the different time steps $1,\ldots,Tx$.
- A sequence model can be used to generate musical values, which are then post-processed into midi music.
- You can use a fairly similar model for tasks ranging from generating dinosaur names to generating original music, with the only major difference being the input fed to the model.
- In Keras, sequence generation involves defining layers with shared weights, which are then repeated for the different time steps $1,\ldots,Tx$.

NLP and Word Embeddings

Word Representations

Word representation

V = [a, aaron, …, zulu, <UNK>]

|V| = 10,000

1-hot representation

Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)

O_{5391} O_{9853}

I want a glass of orange juice.

I want a glass of apple ?.

Andrew Ng

it makes it harder for algorithm to learn the relations, makes it hard for it to generalize, embeddings are distant - no direct relation in any way

Featurized 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						
verb						

300

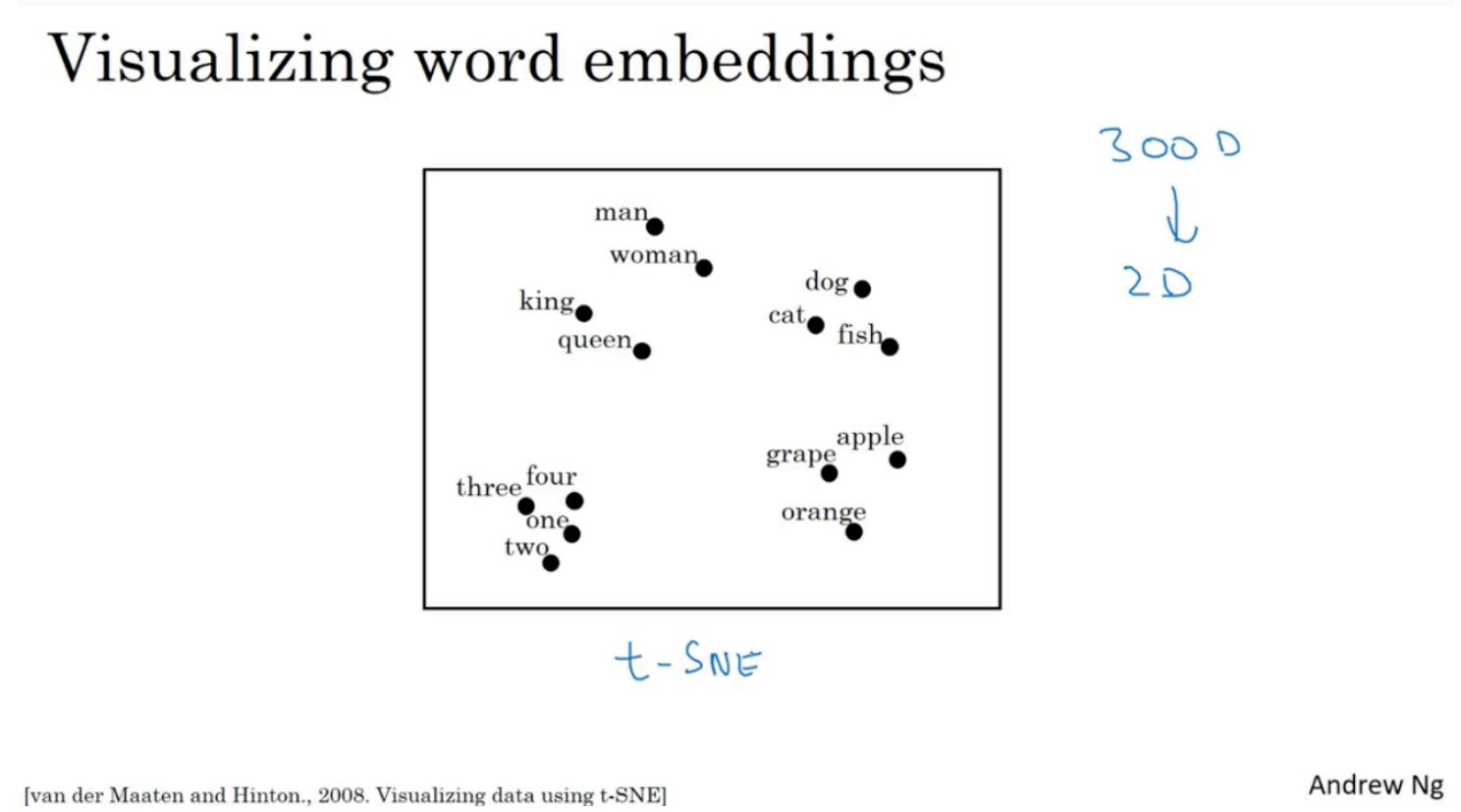
e_{5391} e_{9853}

I want a glass of orange _____.

I want a glass of apple _____.

Andrew Ng

with this approach, representations for orange and apple take on similar values



another general approach is to visualize them by embedding these vectors into 2D, a commonly used algorithm for this task: t-SNE

Using Word Embeddings

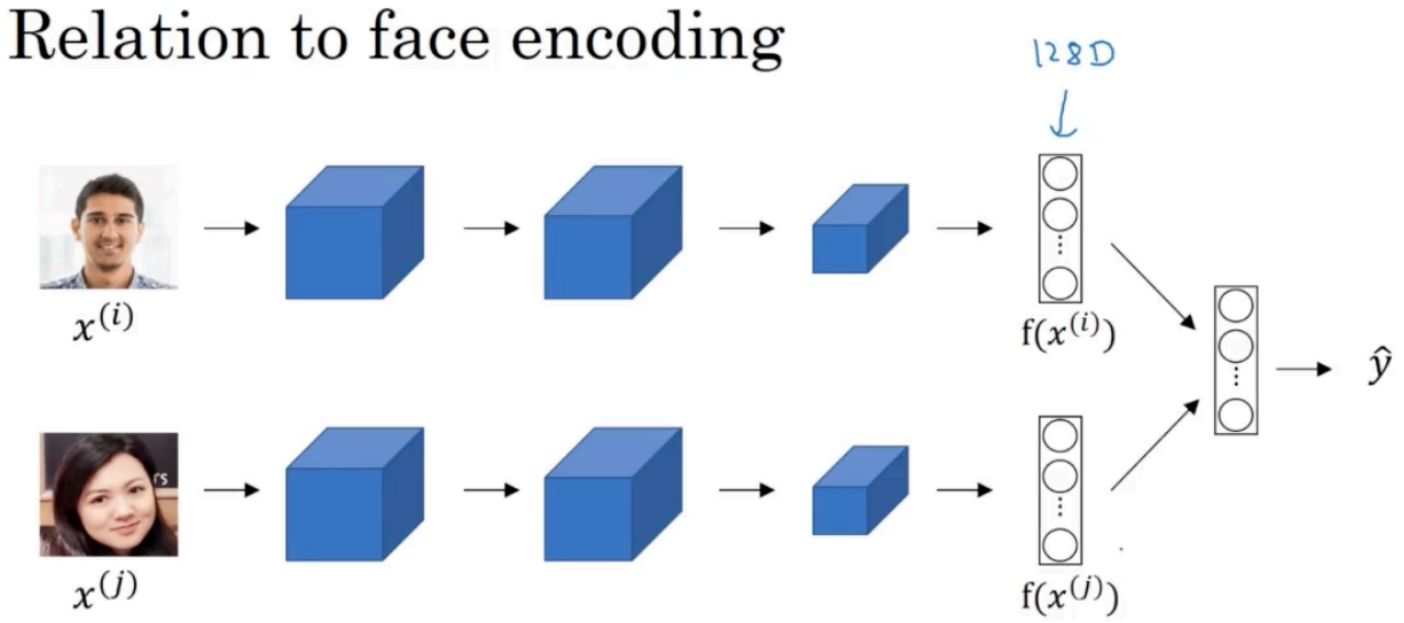
- *Sally Johnson is an orange farmer.*
- 1 1 0 0 0 0

New Input:

- Robert Lin is an *apple farmer*
- Robert Lin is a *durian cultivator*
  may not know the durian if small data, but with word embedding, may even understand and match "durian cultivator" to "orange farmer".
  1B - 100B words
  by reading massive amounts of text
  transfer this knowledge to task like name entity recognition
  (probably will use BRNN for such task)

Transfer Learning and Word Embeddings

1- Learn word embeddings from large text corpus (1-100B words) (Or download pre-trained embedding)
2- Transfer embedding to new task with smaller training set (ex: 100k)
3- (Opt) Continue to finetune the word embeddings with new data

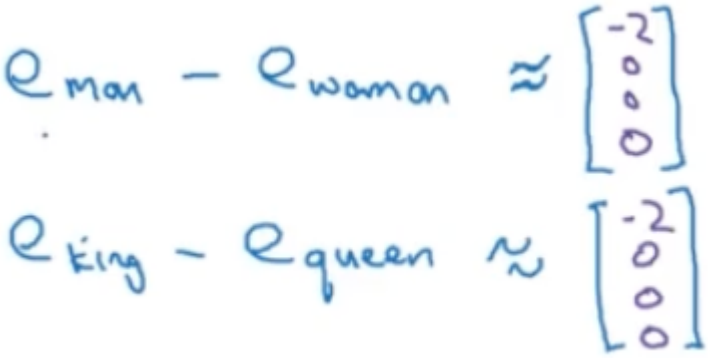


[Taigman et. al., 2014. DeepFace: Closing the gap to human level performance] Andrew Ng

encoding = embedding
encoding: even take in a face that has never seen before, and creates a encoding for it
embedding: have a fixed size vocabulary, embeds sentences and meanings

- difference in between the terms

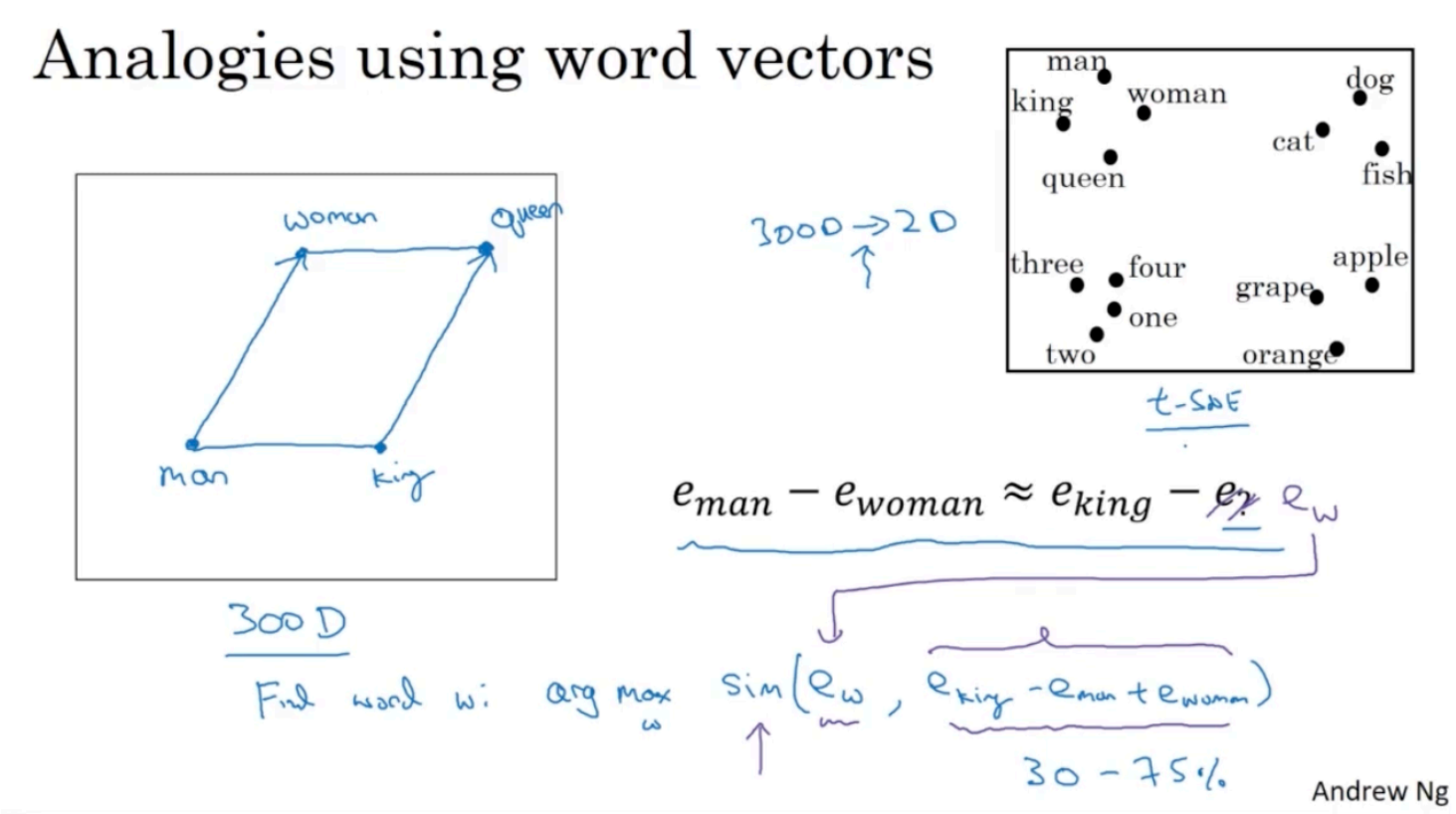
Properties of Word Embeddings
Analogies:
Man -> Woman as King -> ??? - Can the algorithm (embedding) figure this out by itself?



yes
eman - ewoman : try to find a vector that is equivalent to this - that is: eking - equeen

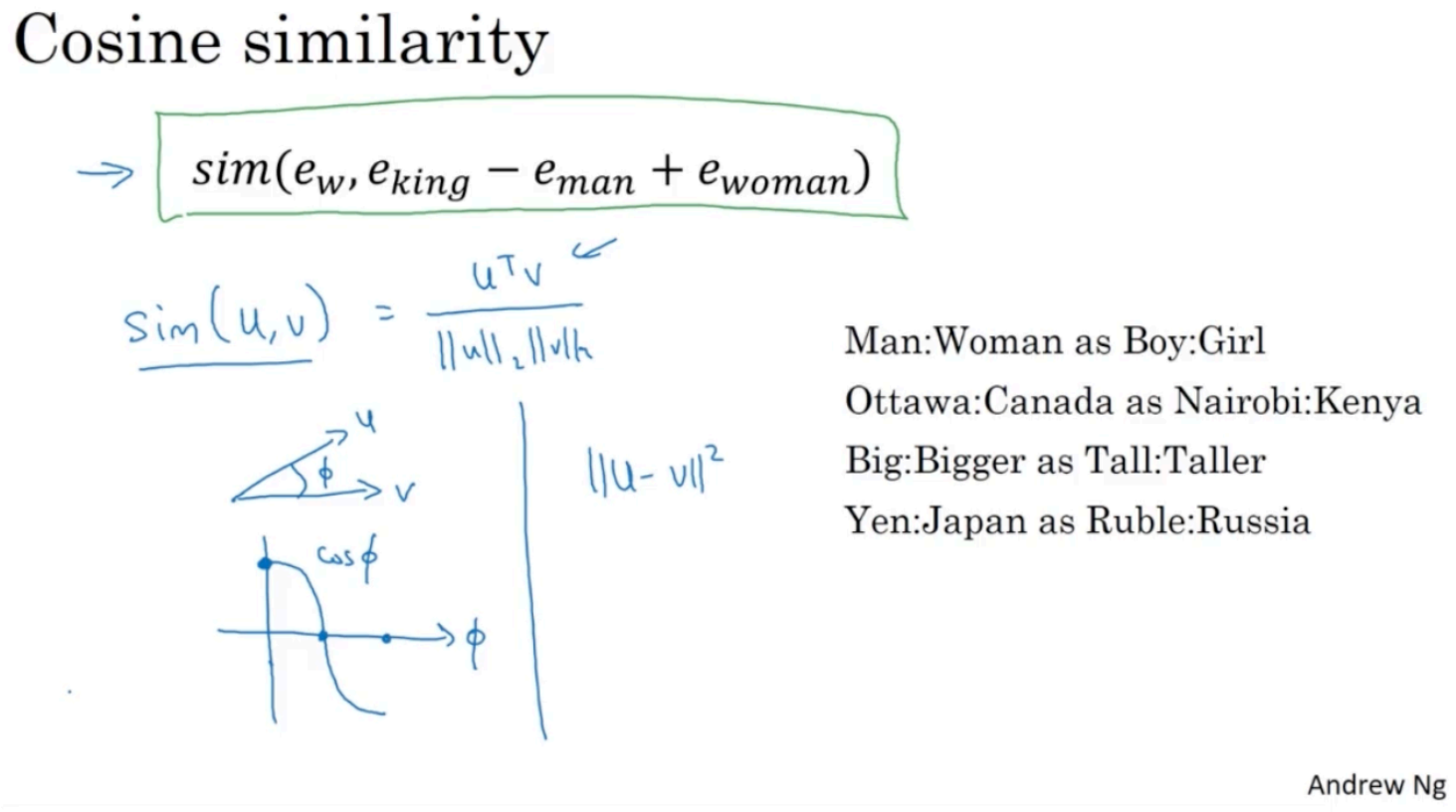
- one of the first surprising findings about word embeddings

t-SAE: takes data and plots non-linearly, there is not relationship as such given in the below example (in most cases)



Similarity Function

most commonly used: Cosine similarity



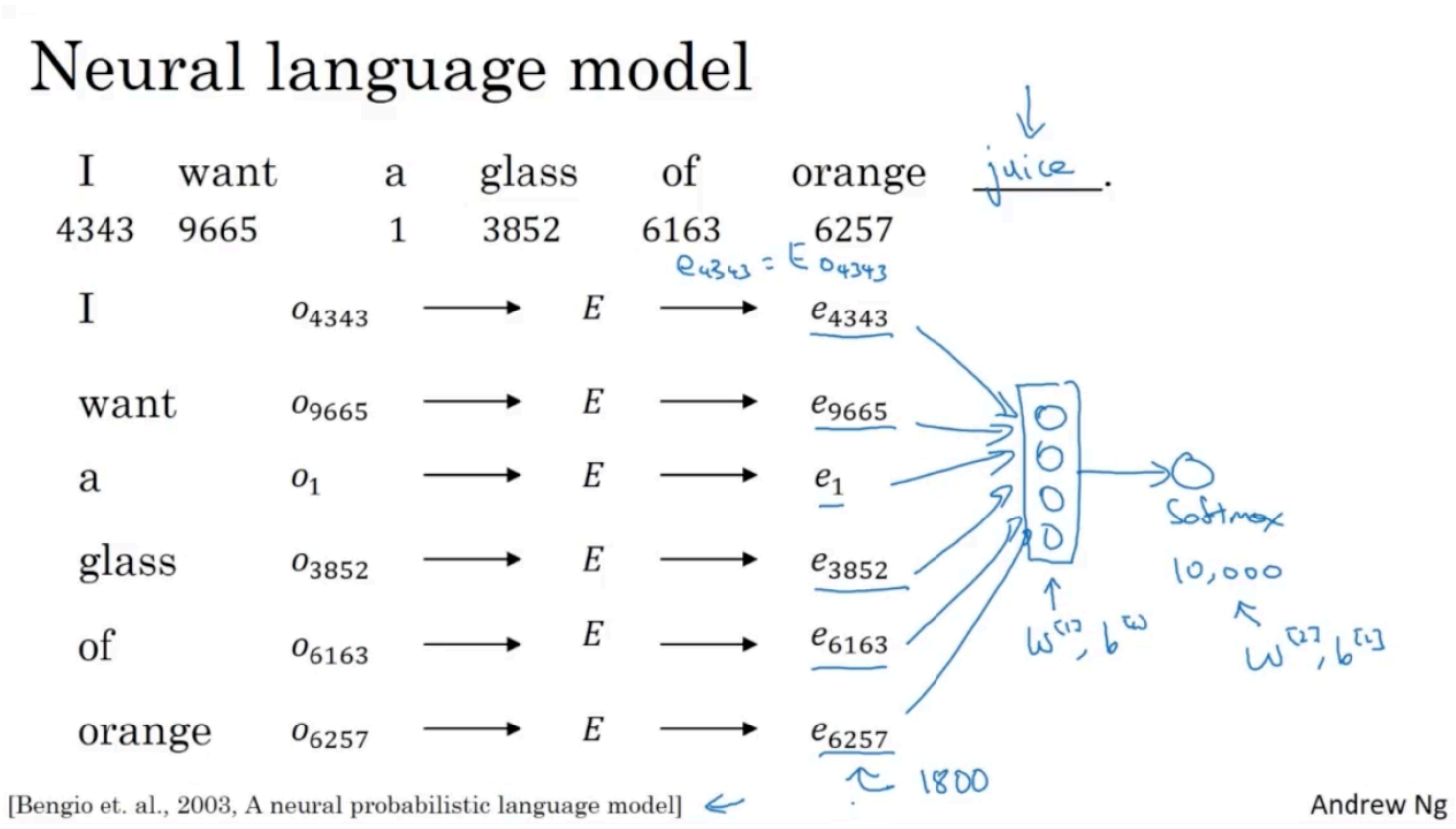
Embedding Matrix

- A matrix learned during word embedding algorithms.
- Dimensions: (embedding dimension) x (vocabulary size). E.g., 300 x 10,000.
- Each column represents the embedding vector for a specific word in the vocabulary.
- **E** multiplied by a word's one-hot vector (e.g., O_{6257} for "Orange") extracts the corresponding embedding vector (e.g., e_{6257}). Because all elements in one-hot vector are 0 except one, this

multiplication gives a column, that represents that specific word, from the E matrix.

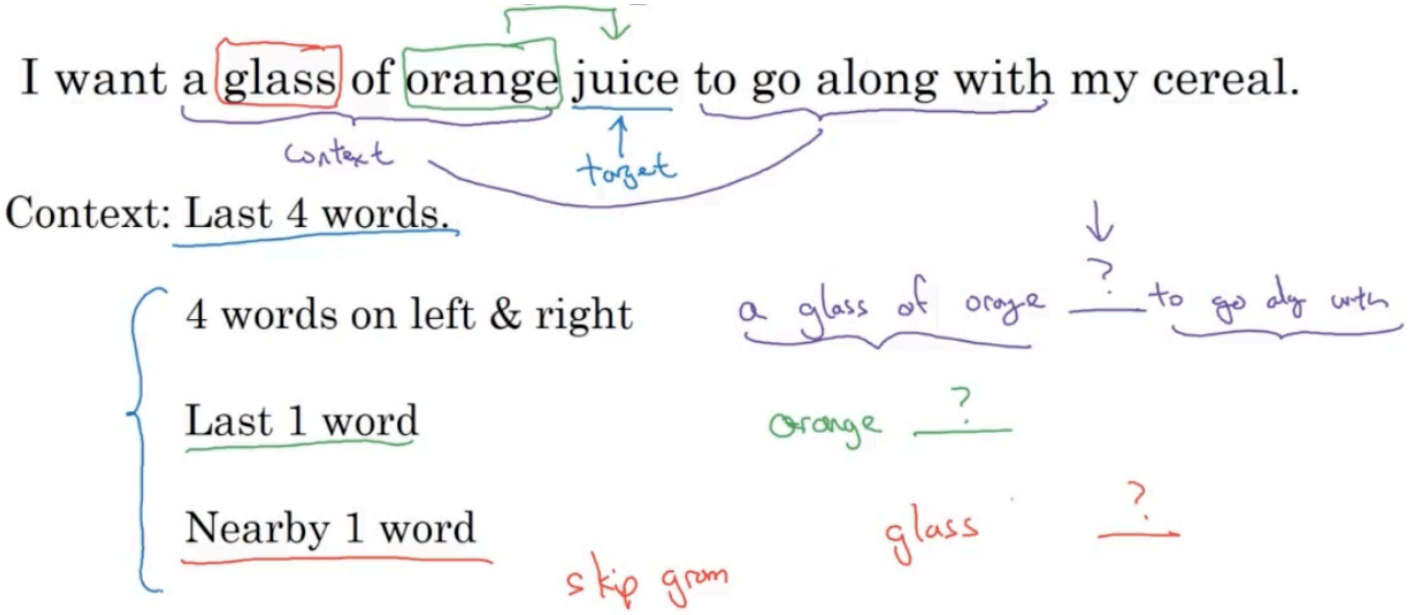
- The result of Multiplying E O_n is equal to E_n.

Word Embeddings



- *6 embedding matrixes stacked together for each word: 1800 dimensions*
- *another hyperparameter, how many words to back will be looked when predicting the output (ex: always 4 words) - making input size fixed -> 1800 becomes 4x300=1200*
  earlier but pretty successful algorithm

Other context/target pairs



if want to build Language Model, general to give the first type of context - if you just want to learn the embeddings, other choices are relevant too.

Word2Vec

simpler, computationally more efficient

Skip-grams

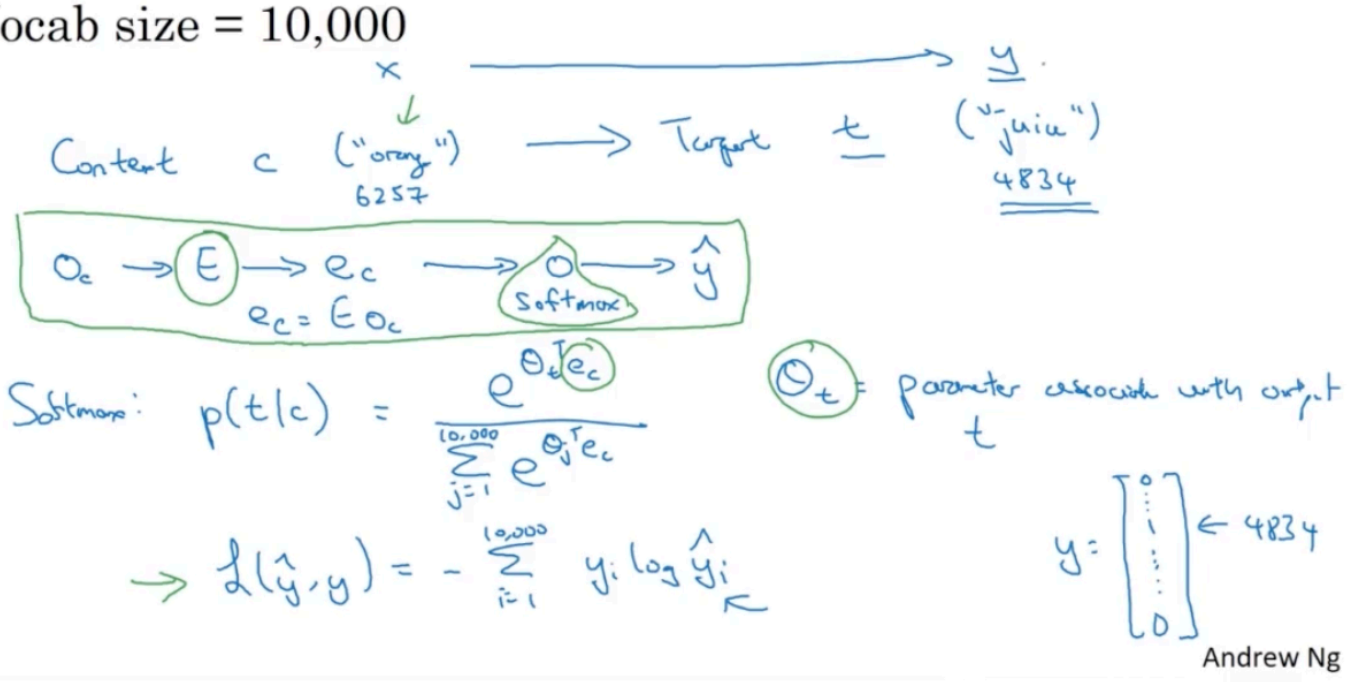
I want a glass of orange juice to go along with my cereal.

Context	Target
orange	juice
orange	glass
orange	my

pick a target within some window (ex: +/- 5)
setup a supervised learning problem
not a easy learning problem - there could be a lot of words

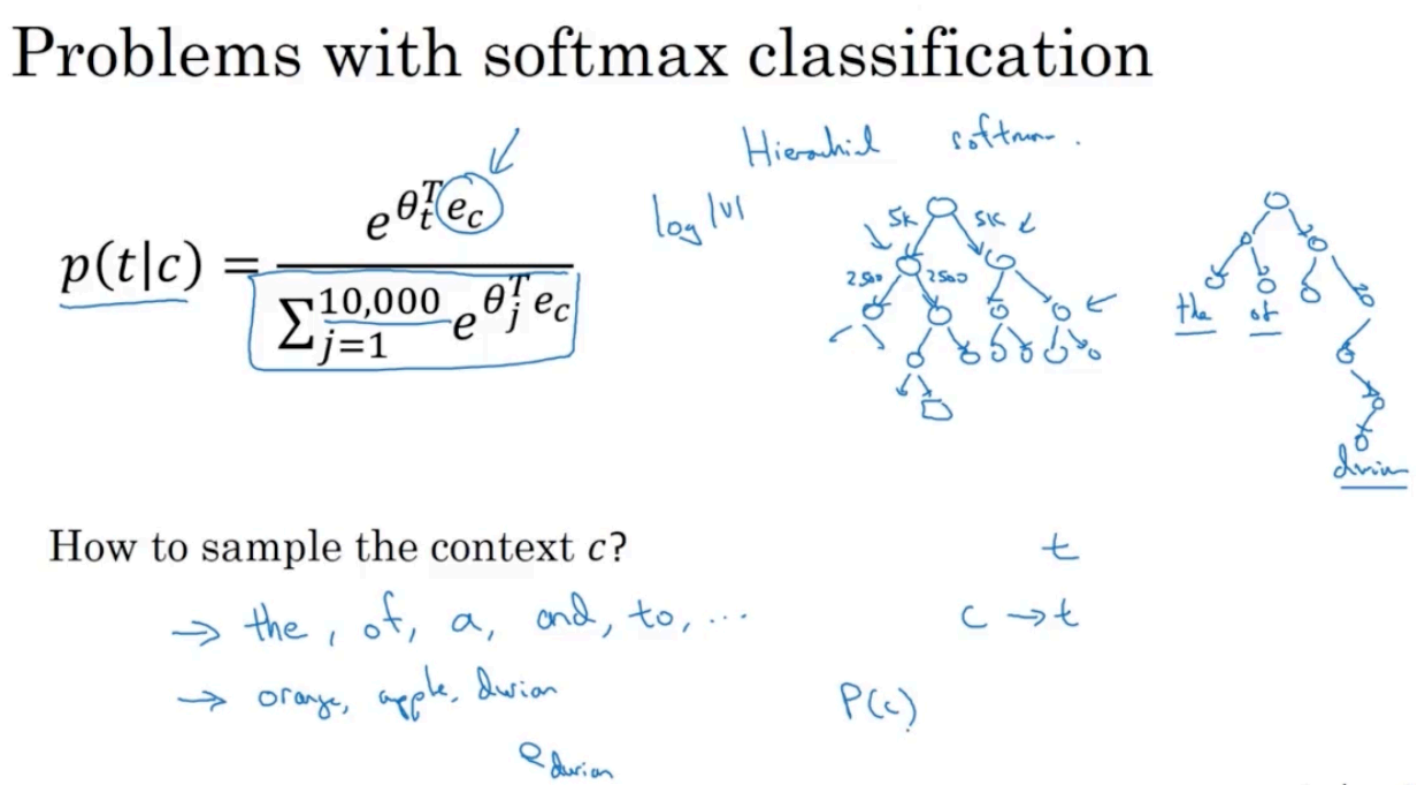
Model

Vocab size = 10,000



Andrew Ng

!

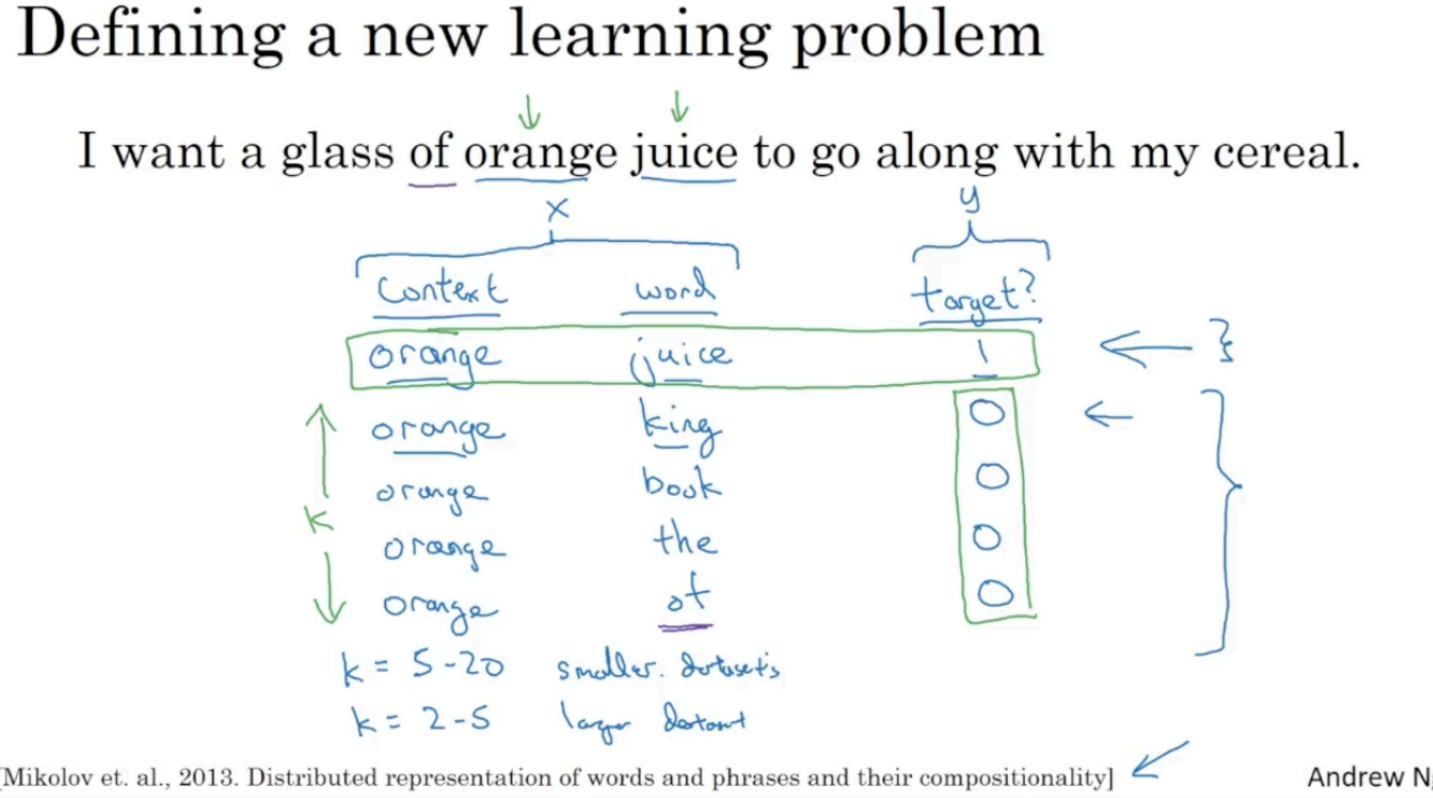


hierarchical softmax: does not use equally weighted trees, less commonly words being buried deep in the tree - do not need to go that deep generally
distribution is not taken entirely uniformly, does not want to update values frequently for words that does not come up frequently - different heuristic options for how to choose

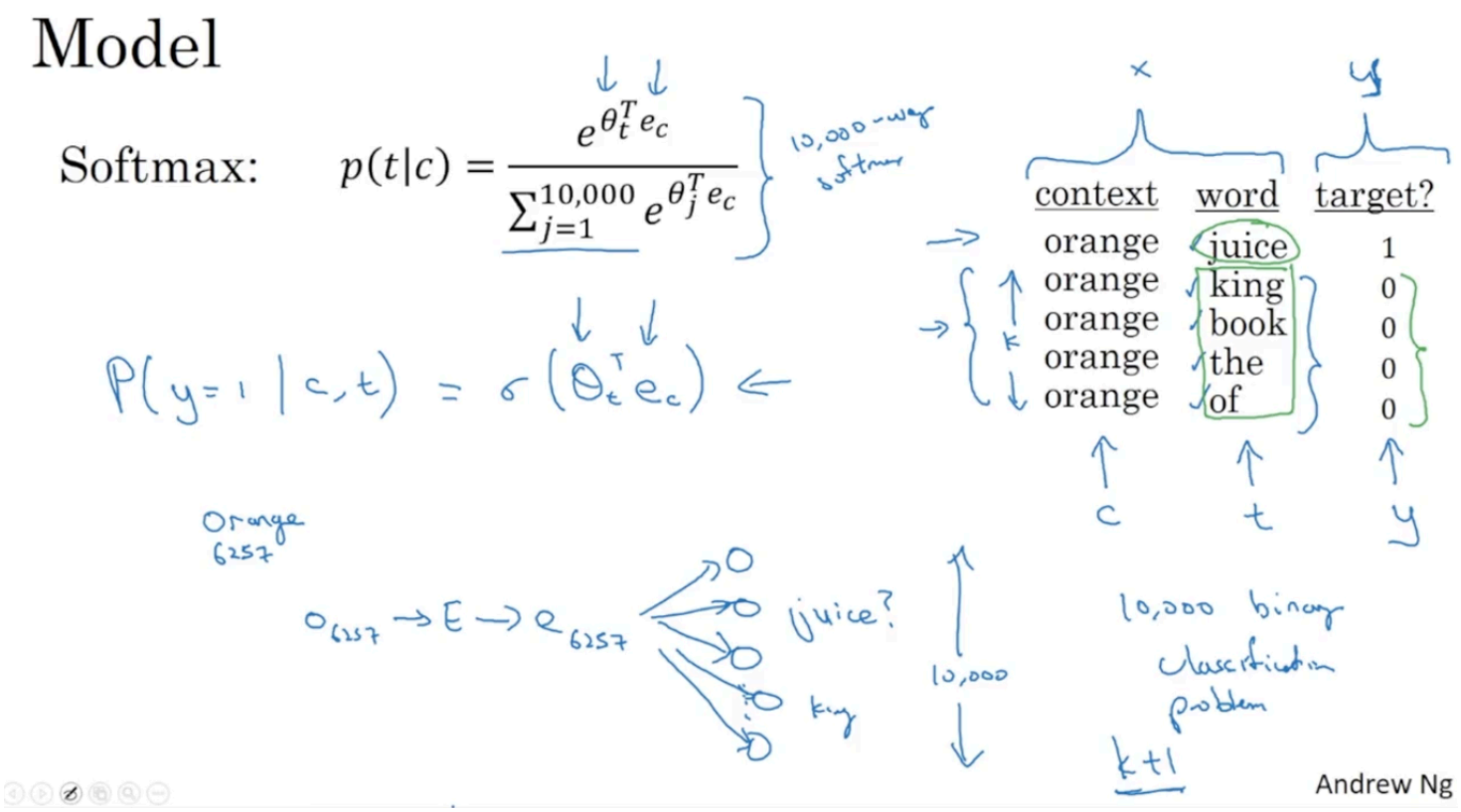
Negative Sampling

something similar to skip-gram model, but much more efficient

give a positive pair, then for some number of examples, say 'k', put in negative examples. Then create a supervised learning problem that inputs x and predicts the target y.

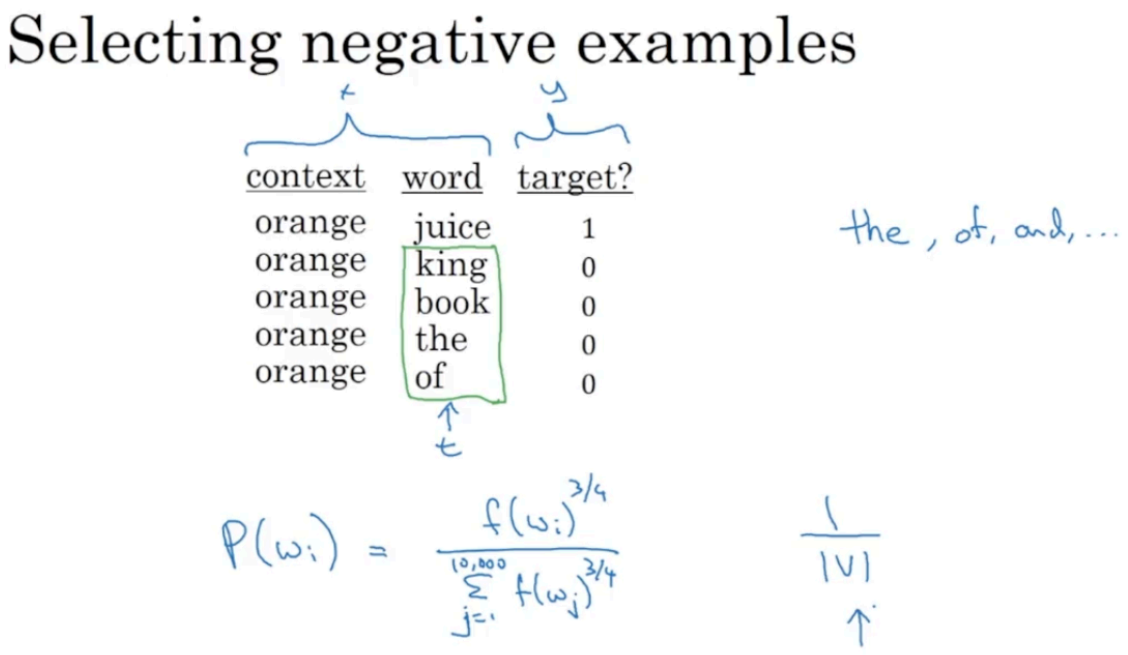


Model Definition



after you choose the word 'orange', how do you sample negative examples?
random - there are words that comes up frequently - *the, of*
soln: take an heuristic value, sample proportional to frequency of the word

- random value but is used: ^(3/4)

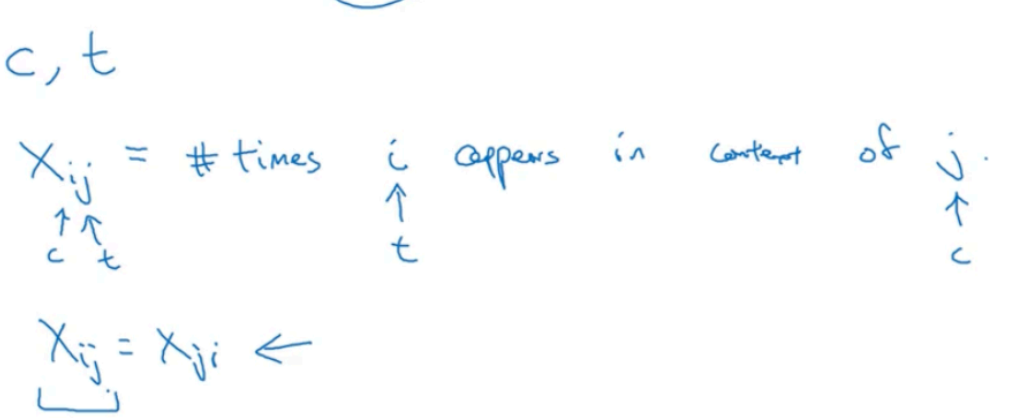


GloVe Word Vectors

not used as much, but has some enthusiast.

GloVe (global vectors for word representation)

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

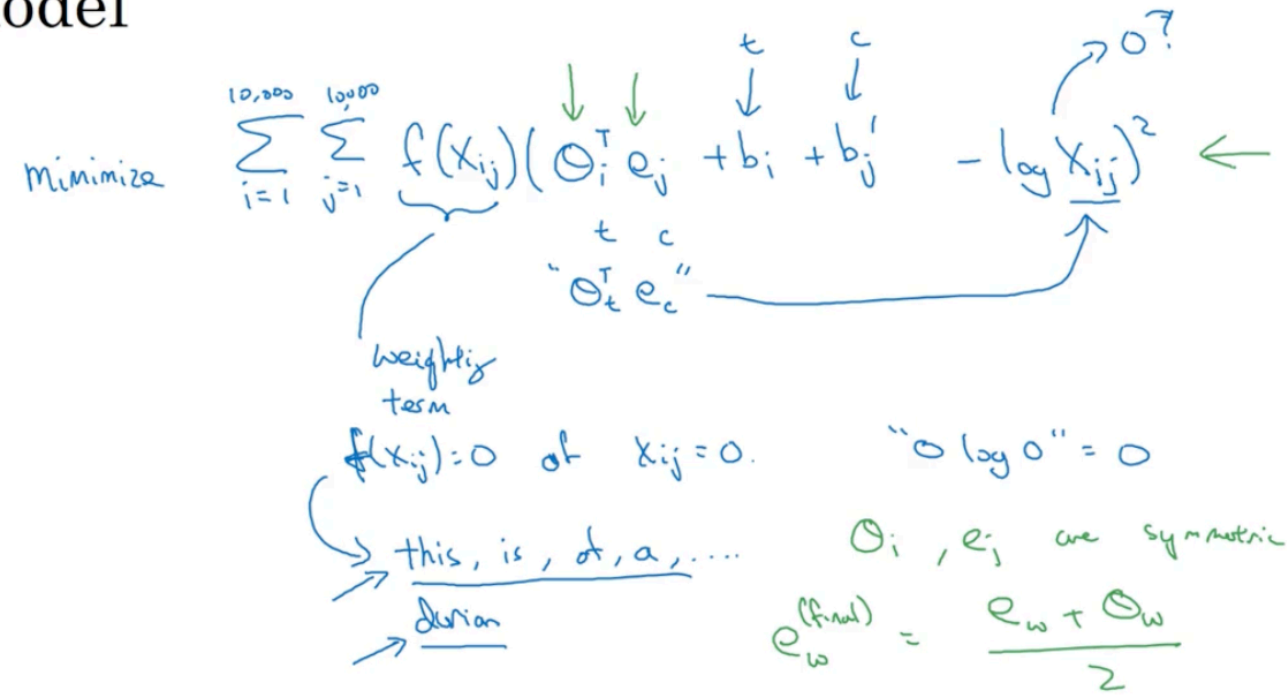


[Pennington et. al., 2014. GloVe: Global vectors for word representation]

Andrew Ng

if equal, they always exist together

Model



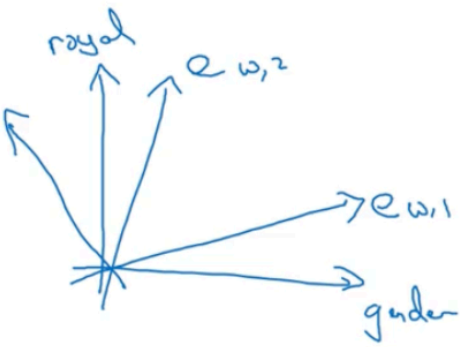
Andrew Ng

simple, but works.

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

A note on the featurization view of word embeddings

	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	←



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

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

Andrew Ng

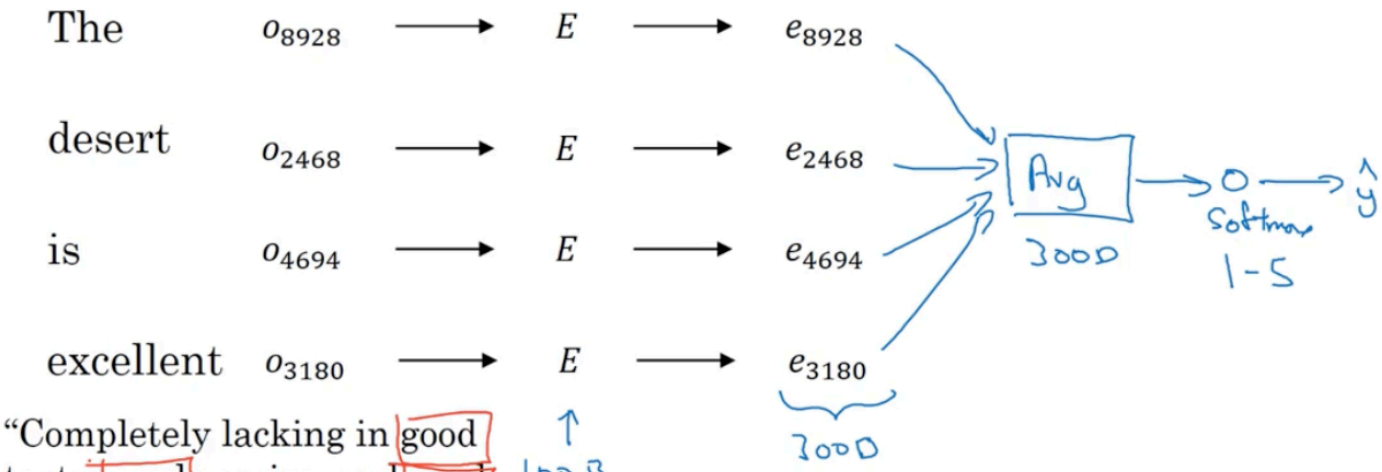
brief proof that with an algorithm like this, axis used by these vectors will be easily interpretable (?)

Sentiment Classification

Simple sentiment classification model

The dessert is excellent ★★★★☆

8928 2468 4694 3180

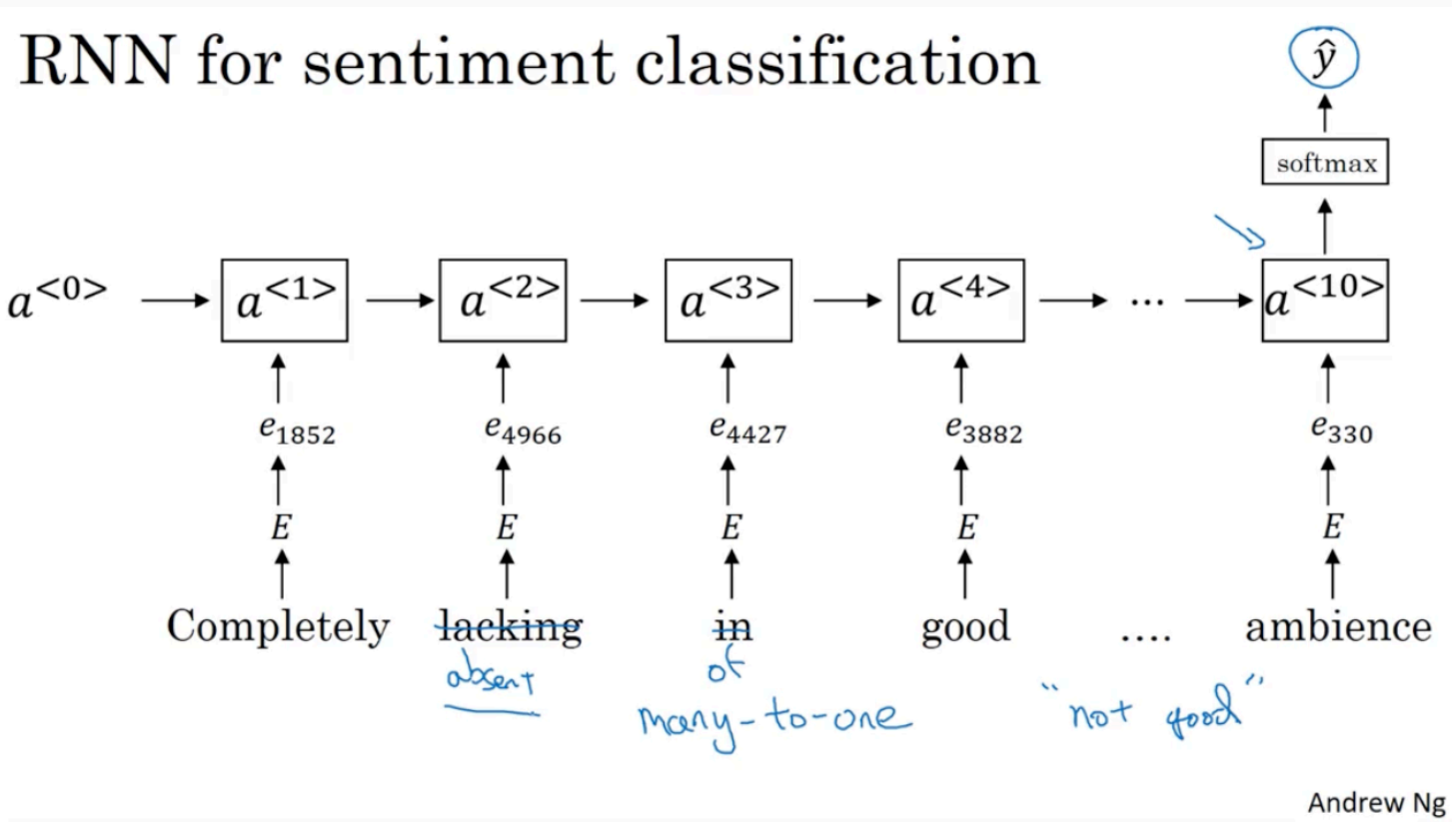


"Completely lacking in good taste, good service, and good ambience."

100B words

Andrew Ng

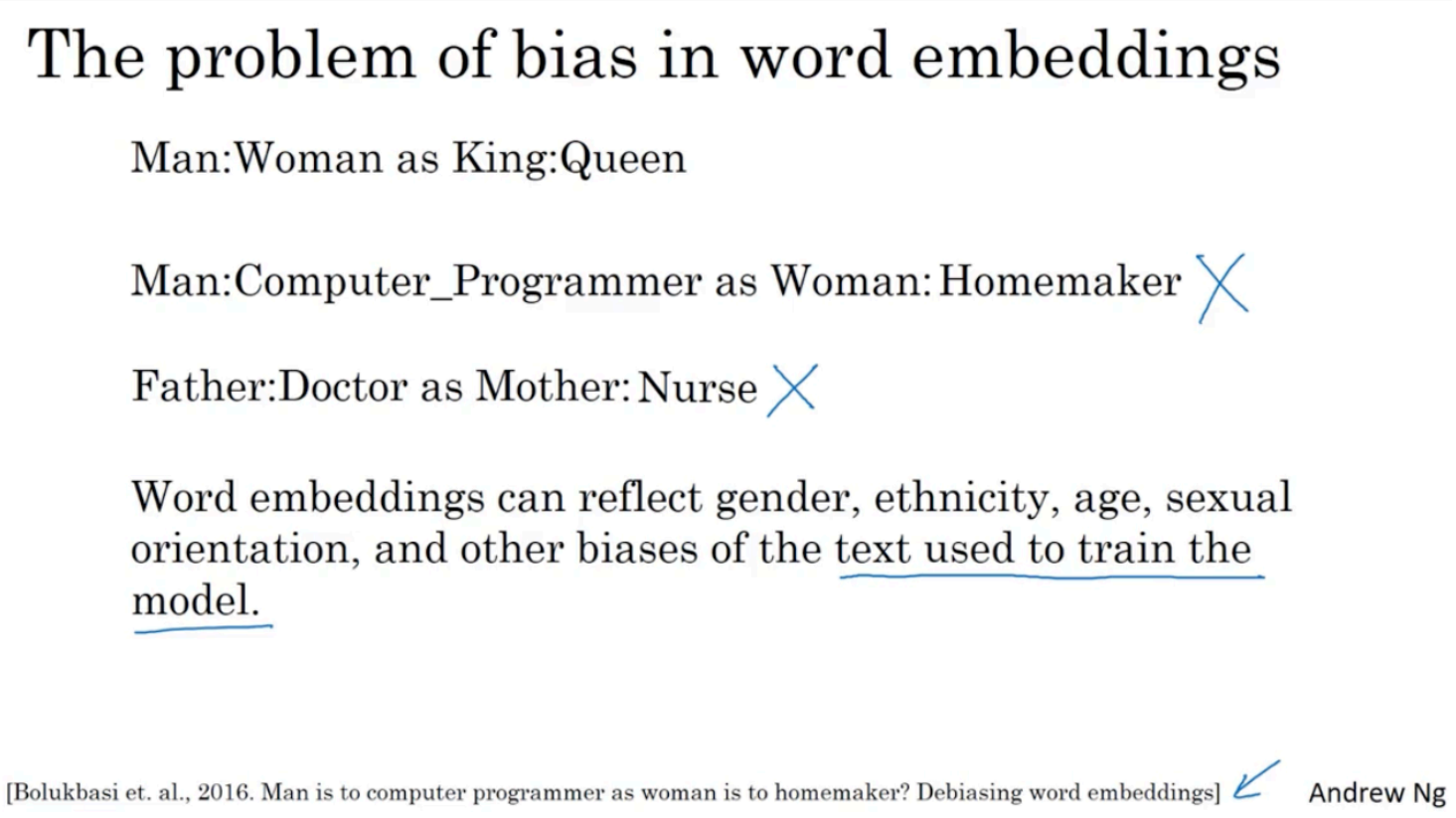
below example becomes a good review even though it is not, soln:

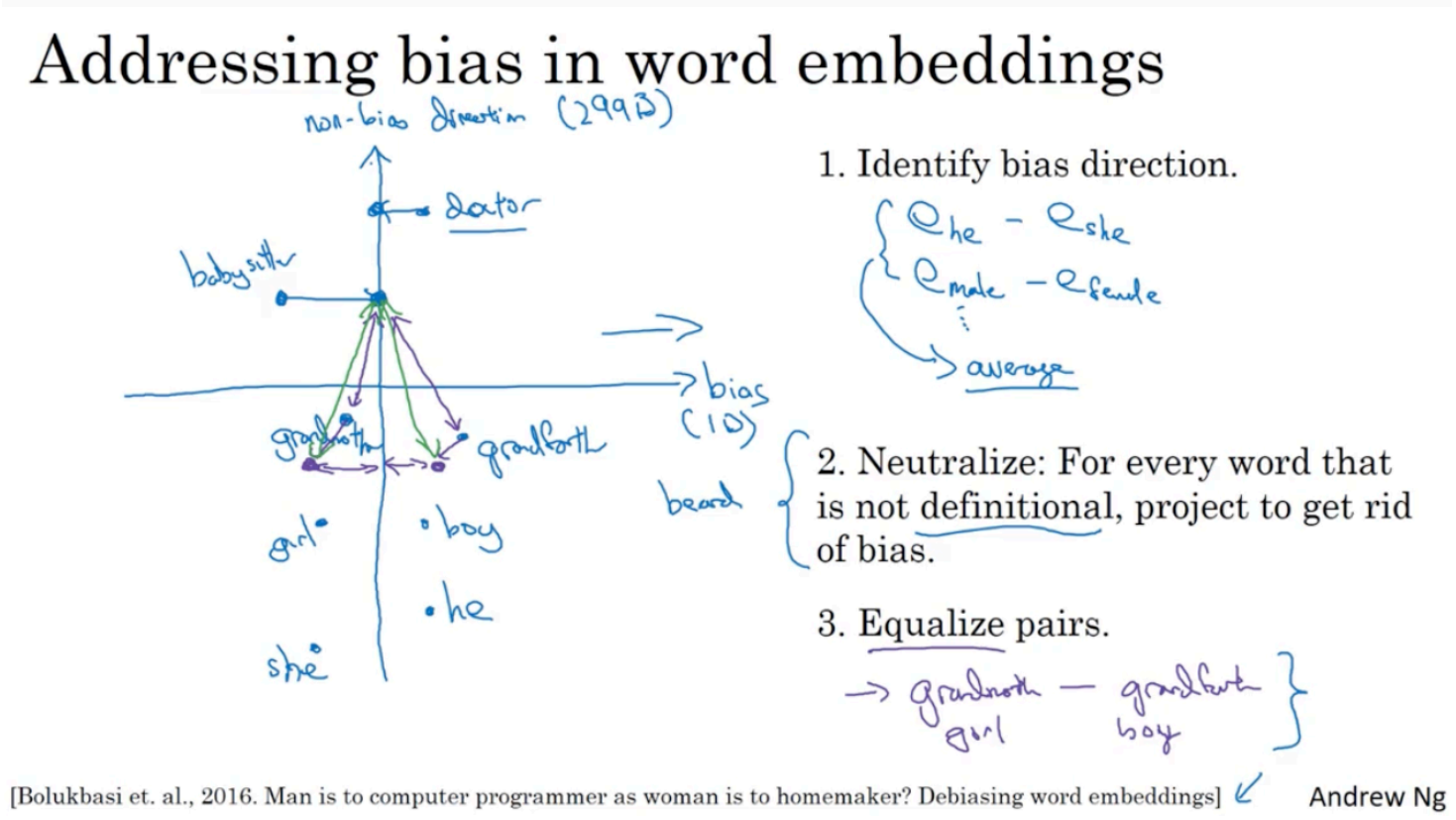


pretty decent sentiment analysis algorithm, better job generalizing, even the words that has not been used specifically

Debiasing Word Embeddings

gender bias, ethnicity bias, sexual orientation bias... - how to diminish





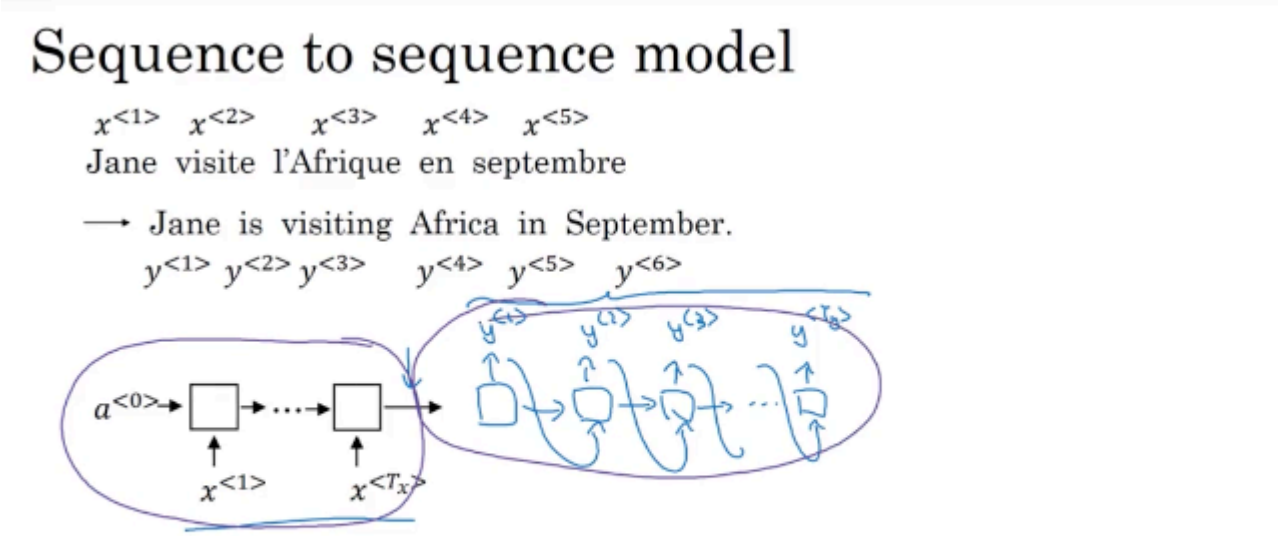
What you should remember:

- Cosine similarity is a good way to compare the similarity between pairs of word vectors.
    - Note that L2 (Euclidean) distance also works.
- For NLP applications, using a pre-trained set of word vectors is often a great way to get started.
- Even with a mere 127 training examples, you can get a reasonably good model for Emojifying.
    - This is due to the generalization power word vectors gives you.
- Emojify-V1 will perform poorly on sentences such as *"This movie is not good and not enjoyable"*
    - It doesn't understand combinations of words.
    - It just averages all the words' embedding vectors together, without considering the ordering of words.
- If you have an NLP task where the training set is small, using word embeddings can help your algorithm significantly.
- Word embeddings allow your model to work on words in the test set that may not even appear in the training set.
- Training sequence models in Keras (and in most other deep learning frameworks) requires a few important details:
    - To use mini-batches, the sequences need to be **padded** so that all the examples in a mini-batch have the **same length**.
    - An `Embedding()` layer can be initialized with pretrained values.
        - These values can be either fixed or trained further on your dataset.
        - If however your labeled dataset is small, it's usually not worth trying to train a large pre-trained set of embeddings.
    - `LSTM()` has a flag called `return_sequences` to decide if you would like to return all hidden states or only the last one.

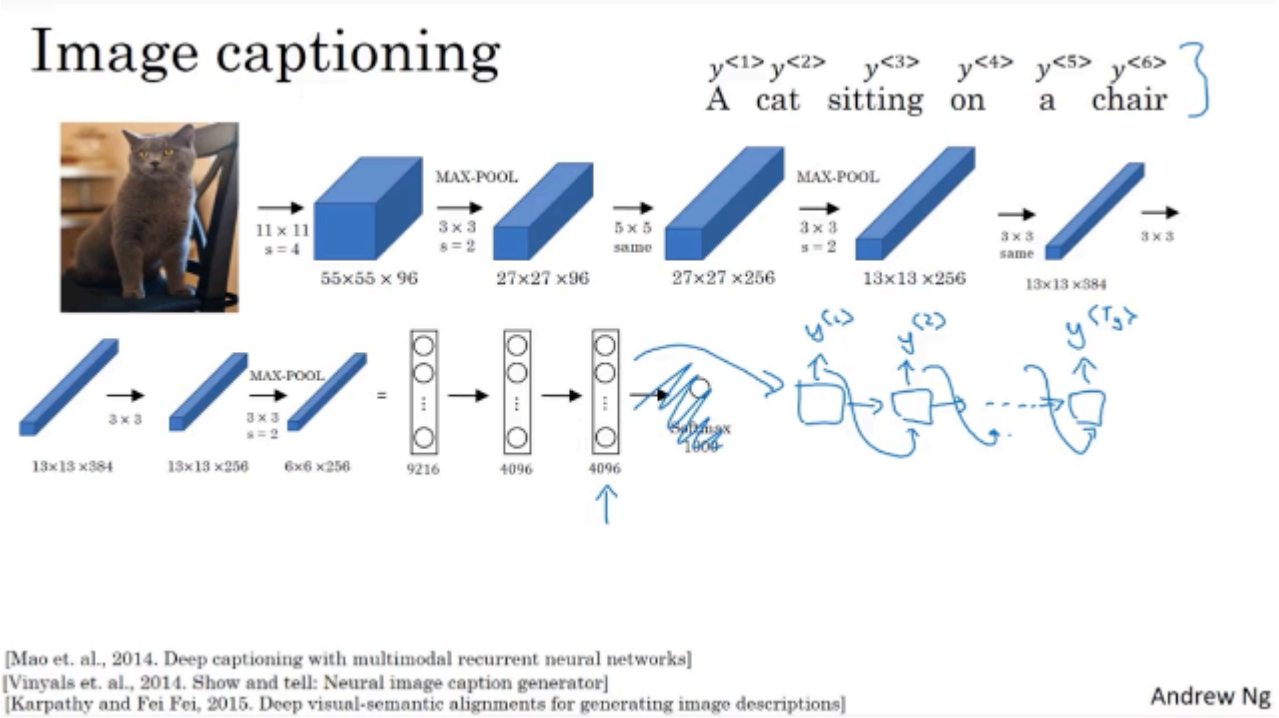
- You can use `Dropout()` right after `LSTM()` to regularize your network.

Various Sequence To Sequence Architectures

Basic Models



[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]
Andrew Ng

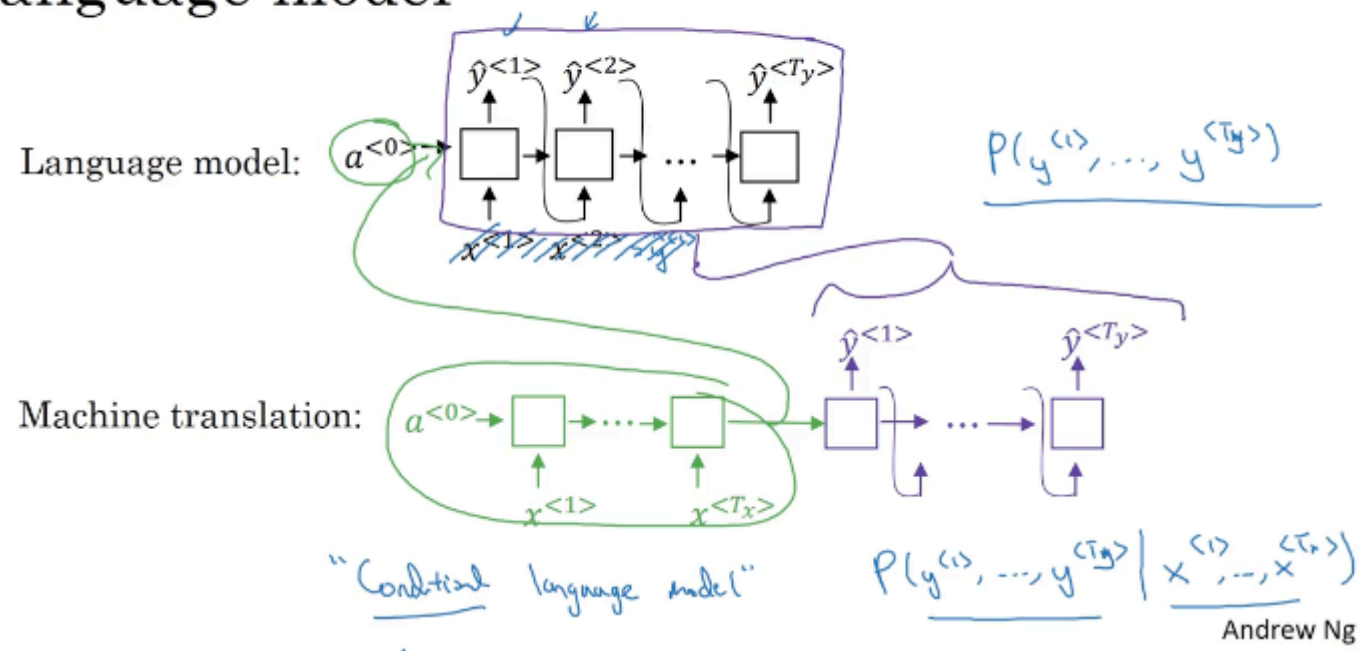


[Mao et. al., 2014. Deep captioning with multimodal recurrent neural networks]
[Vinyals et. al., 2014. Show and tell: Neural image caption generator]
[Karpathy and Fei Fei, 2015. Deep visual-semantic alignments for generating image descriptions]
Andrew Ng

works well especially if caption wanted is not that long

- there were multiple groups that were working on similar things at the time

Machine translation as building a conditional language model



understand the part which makes it 'conditional'

Finding the most likely translation

Jane visite l'Afrique en septembre.

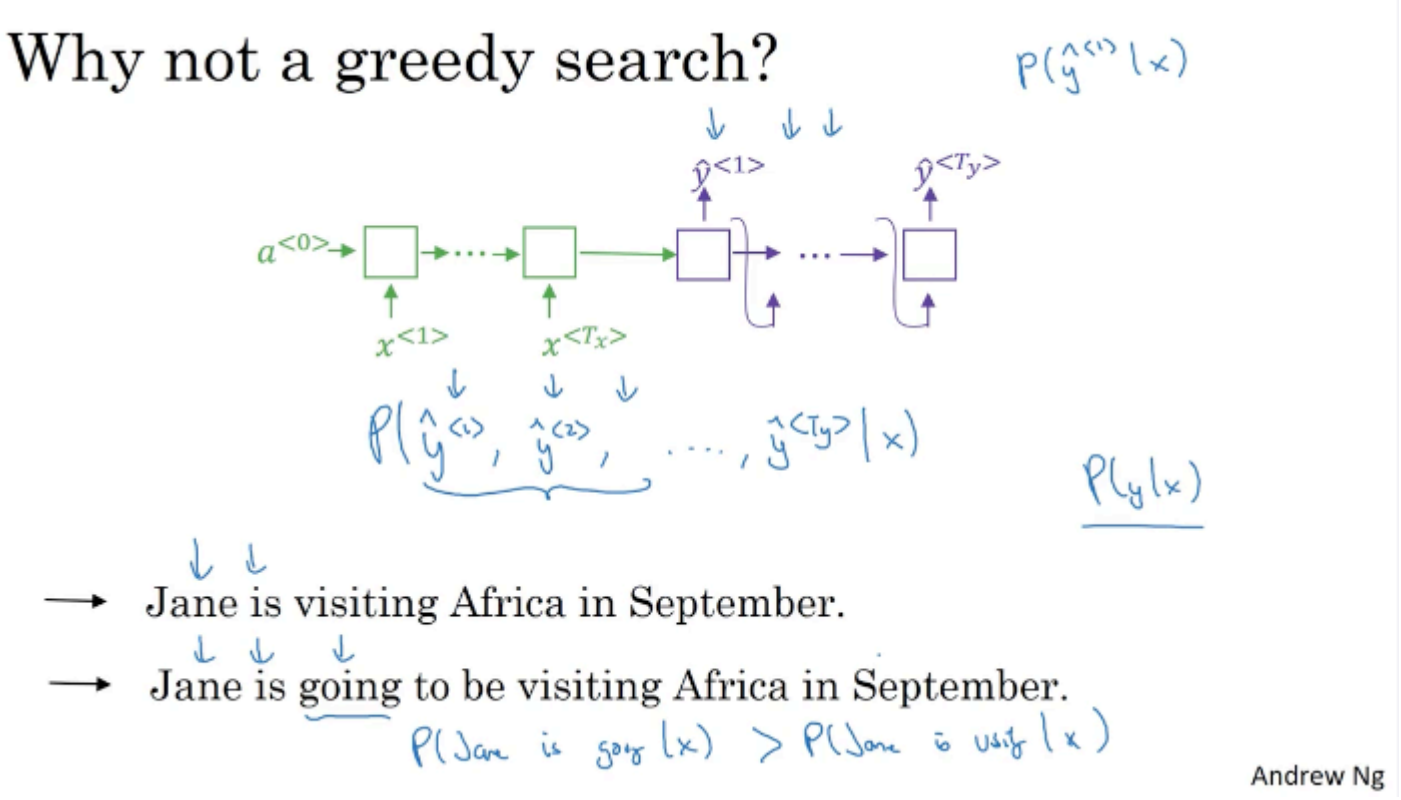
$$P(y^{<1>}, \ldots, y^{<T_y>} \mid x)$$

- 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.

$$\arg\max_{y^{<1>},\ldots,y^{<T_y>}} P(y^{<1>}, \ldots, y^{<T_y>} \mid x)$$

Andrew Ng

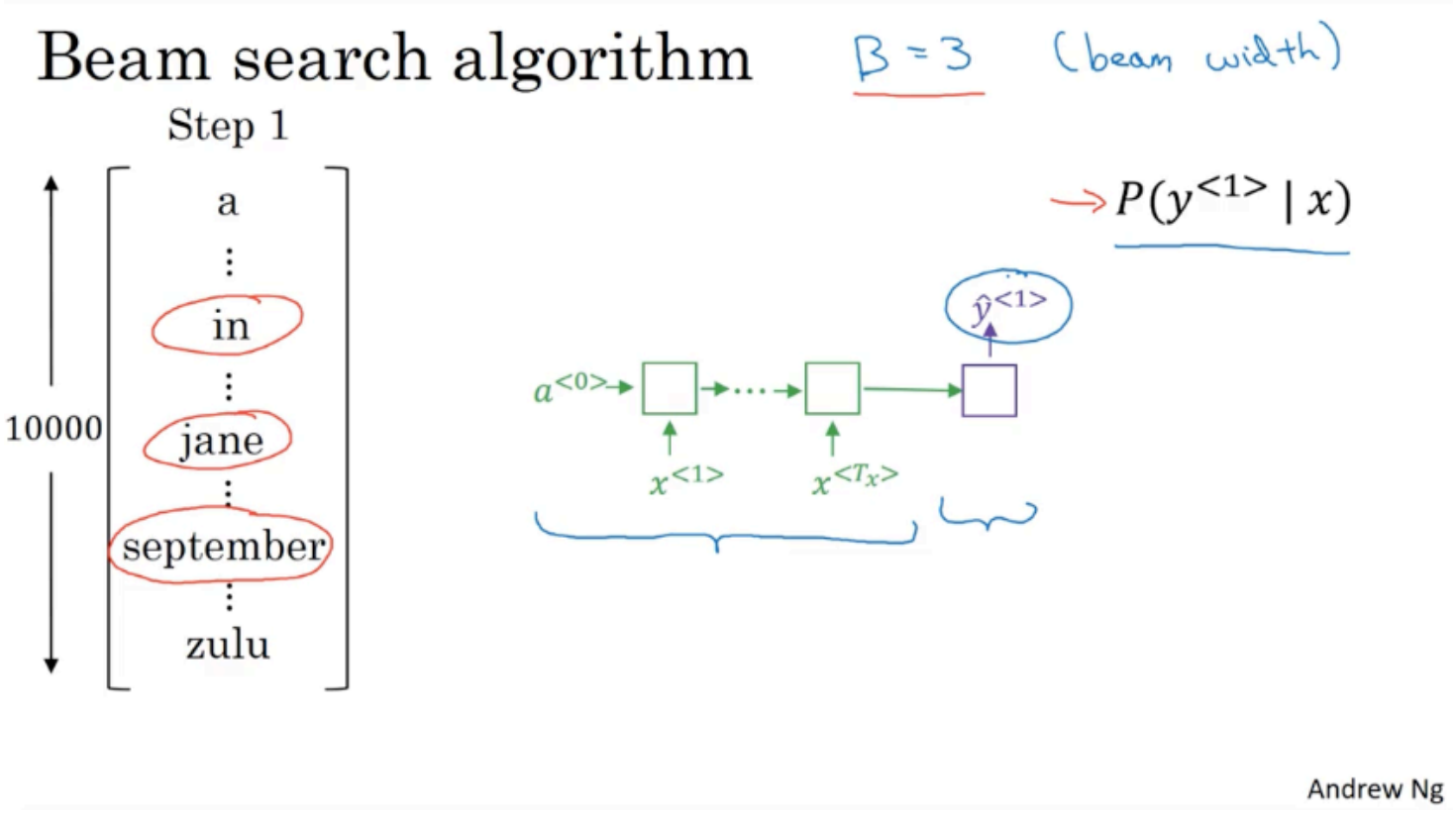
find an english sentence y that maximizes, we do not want to sample it.



- *Pick best word for each unit (highest probability) - greedy search -> does not really work*
- *going is a more general word, so for example, program may say that second sentence, for the third word, 'going' is the most likely choice - which ultimately results in a worse sentence for translation*

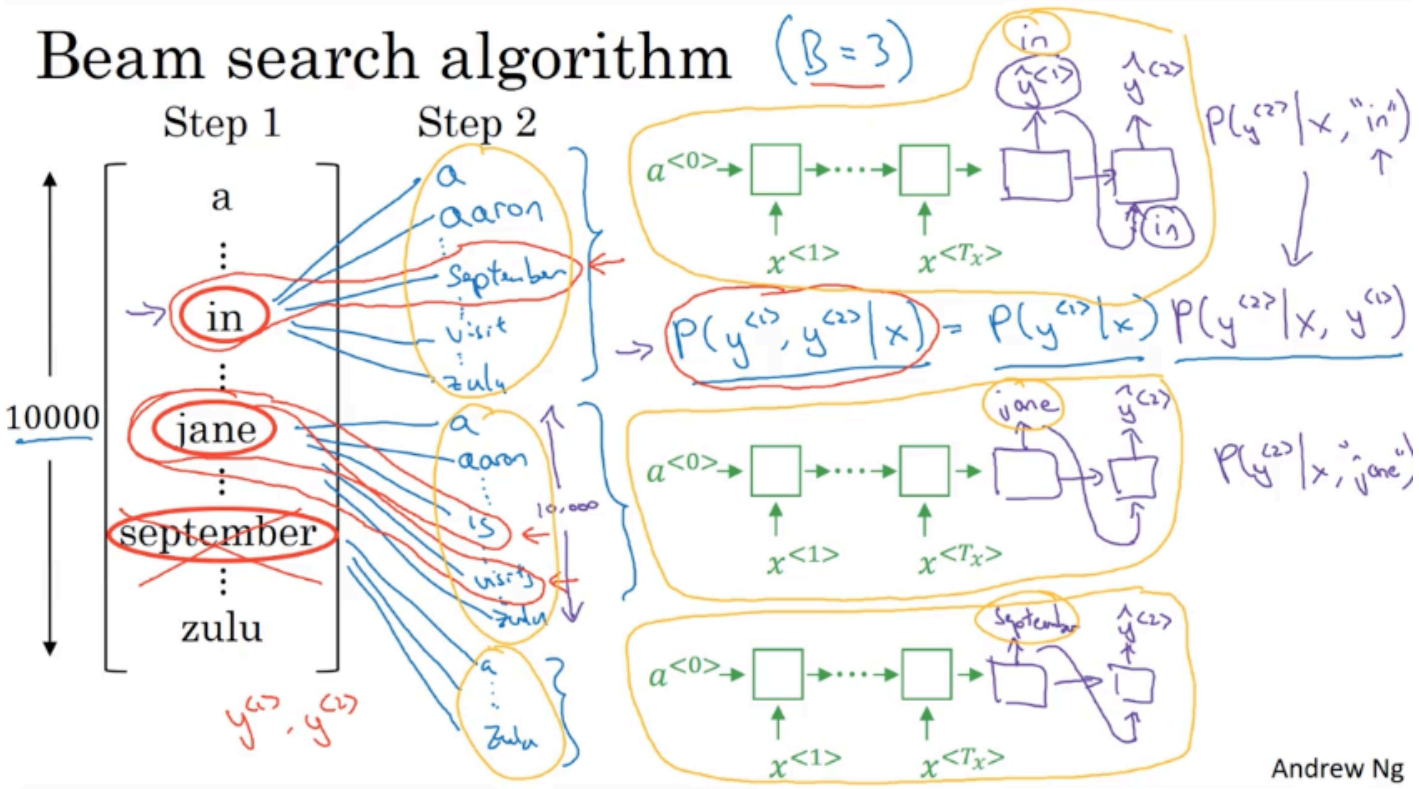
There are numerous possible sentences, for example for 10 word sentences, so we use approximate search algorithms (10k vocab, 10 words = 10k ^ 10 possible sentences)

Beam Search

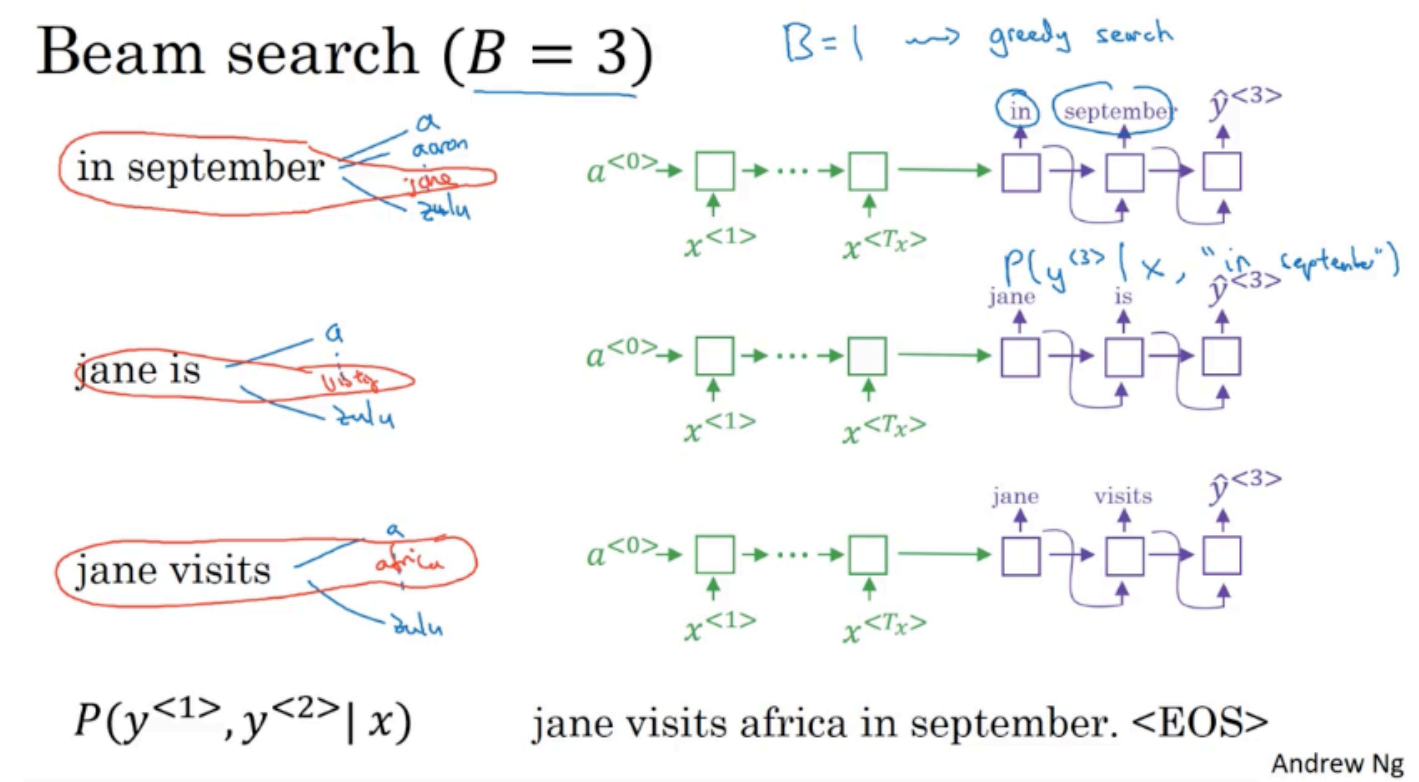


- try to pick the first word in translated sentence

- has parameter B - beam width - can search for several possibilities at once, different than other search algorithms (let's say finds 3 words, chooses them to continue to compute them - hold each for the first word)



- now for each of these choices, find the best second choice, dependent on the first choice
- during iteration, eliminates some choices of branches
- don't instantiate all possible outputs, just choose a number of them



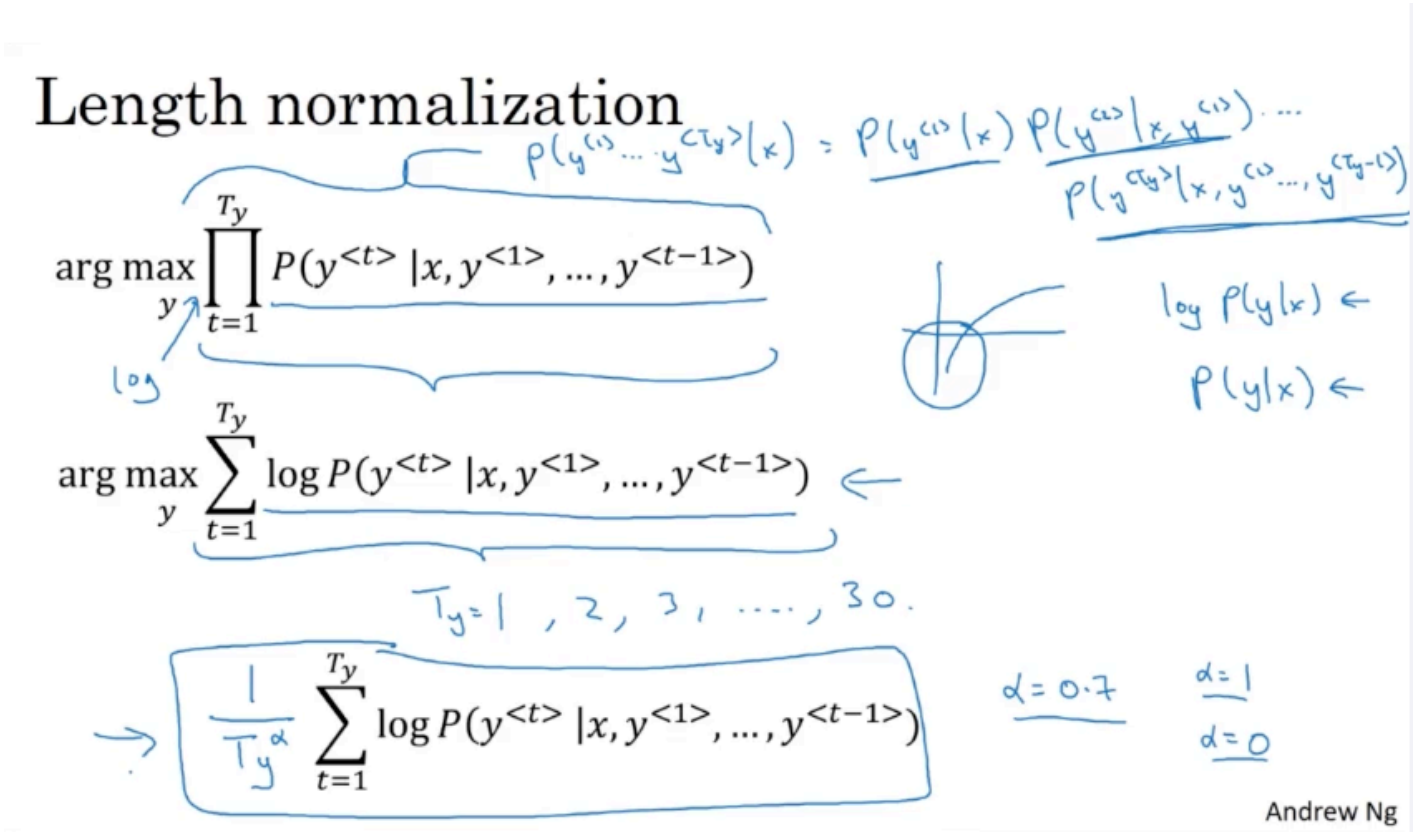
- and iterate

Refinements to Beam Search

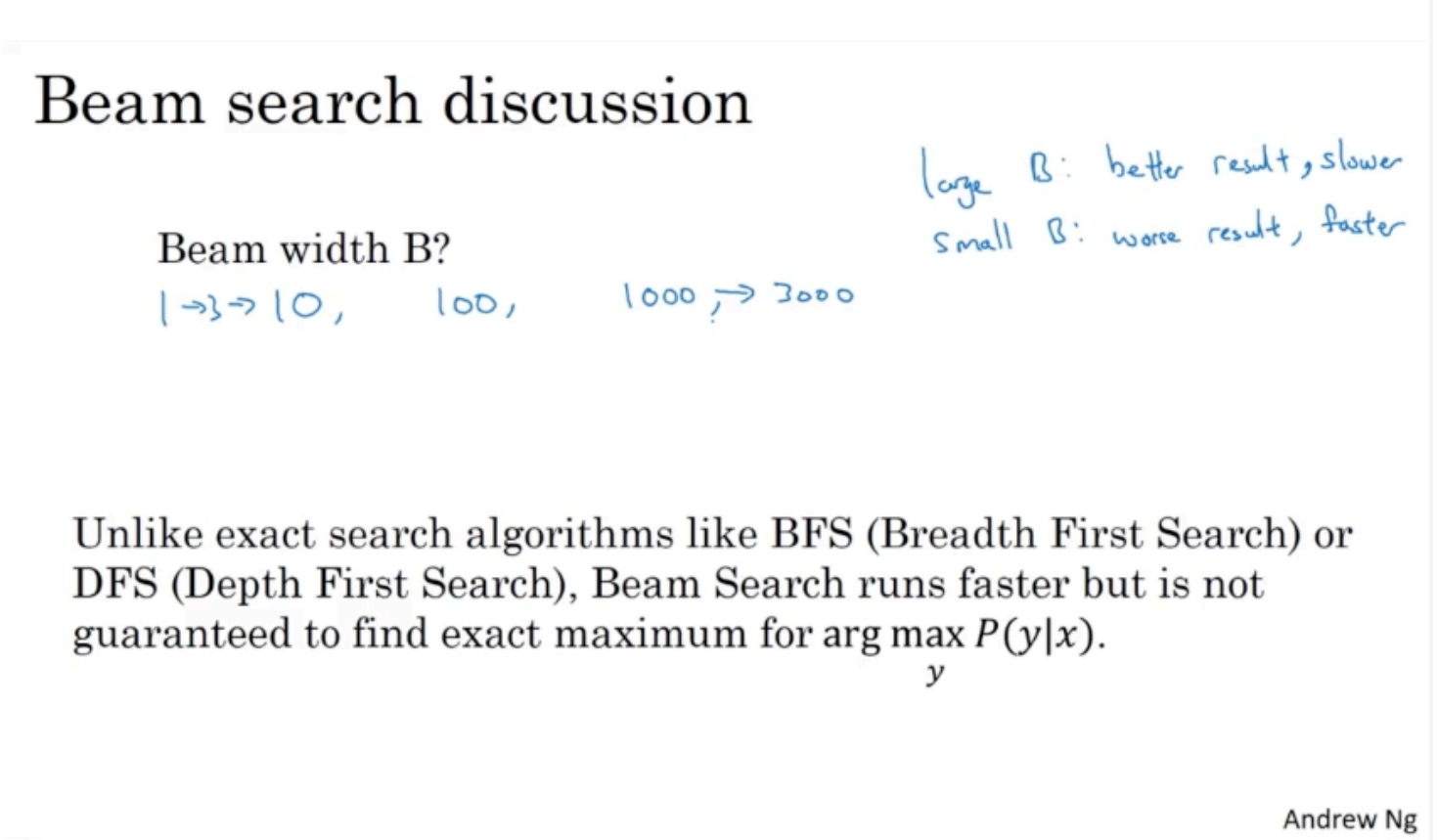
Length Normalization

may get a *numerical underflow* - multiplying many numbers that are very small: 0.0001

- soln: take logs, more stable output values.
- another change to make it work even better: long sentence -> probability is very low. normalize it by the number of words in the sentence, this significantly reduces the penalty. also there are different/softer adjustments (ex: adding another hyperparameter, alpha).



How Big Value For Beam Width B?



Error Analysis in Beam Search

This is an heuristic algorithm, so it does not always find the most optimal/accurate result, so how to apply error analysis here?

- got a result (bad), who is to blame? increase the *B*, never hurts. -> might not solve. When to decide take time to try to make the RNN part better?
- compute both probabilities, p* and p^, both can be bigger than other, dependent on the result you may understand the problem better.

Example

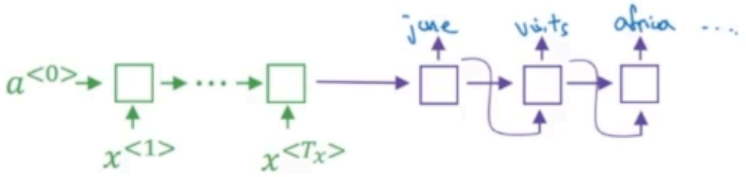
→ RNN
→ Beam Search
B↑

Jane visite l'Afrique en septembre.

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

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

RNN computes $P(y^*|x)$ > ≤ $P(\hat{y}|x)$



Andrew Ng

Logic:

Error analysis on beam search

$P(y^*|x)$
$P(\hat{y}|x)$

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

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

Case 1: $P(y^*|x) > P(\hat{y}|x)$ ⇐

$\arg\max_y P(y|x)$

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

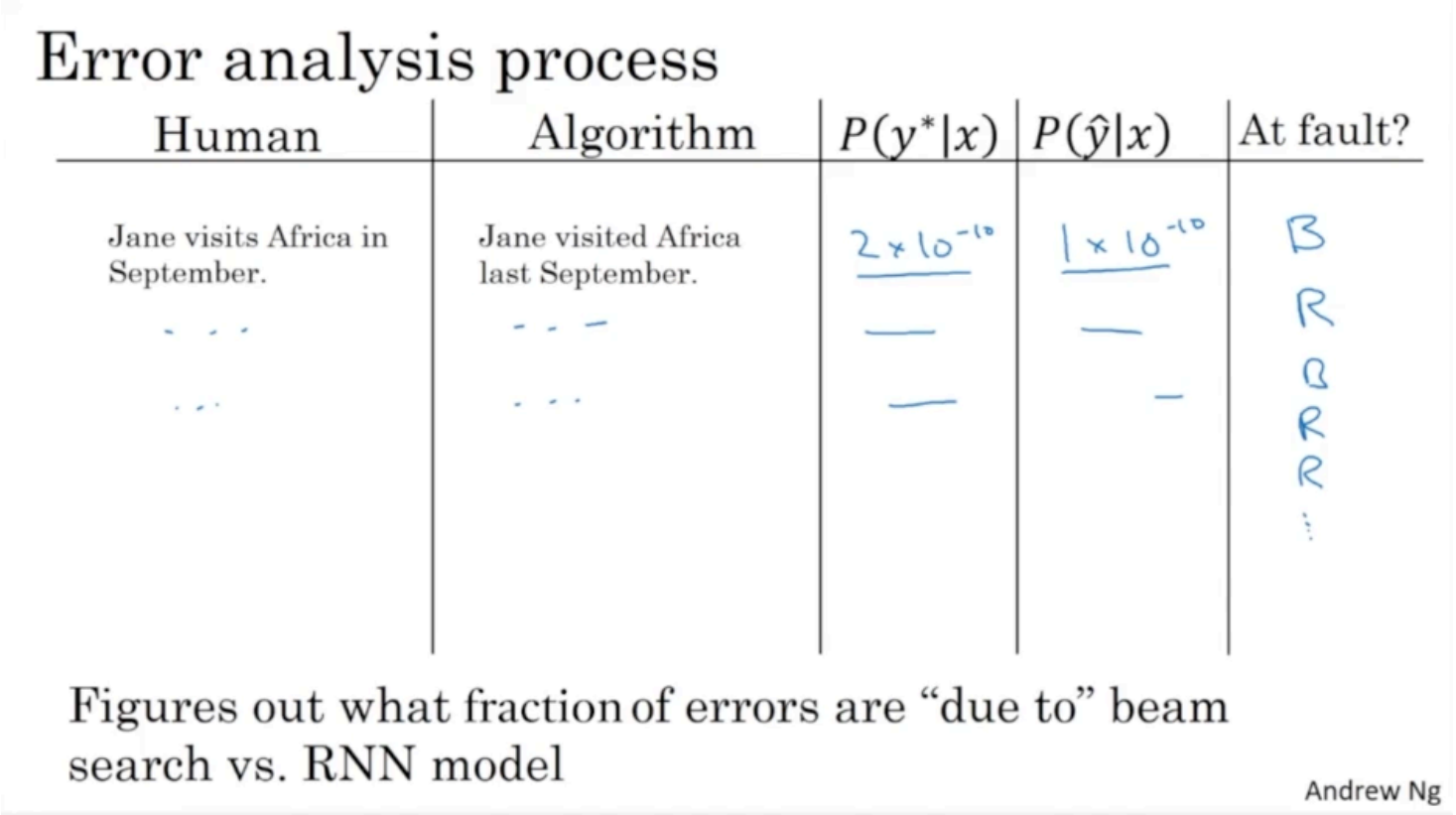
Conclusion: Beam search is at fault.

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

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

Conclusion: RNN model is at fault.

Andrew Ng



Bleu Score

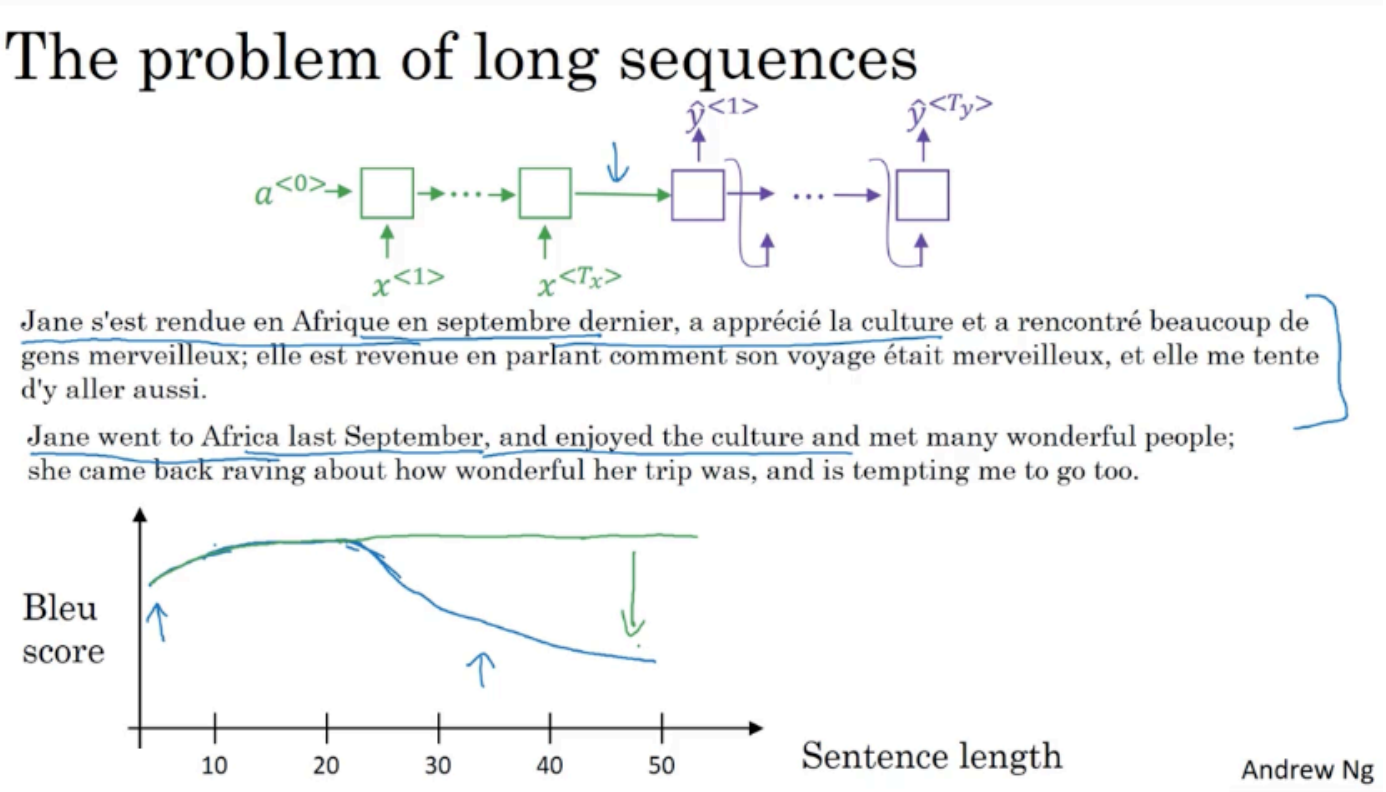
The BLEU (Bilingual Evaluation Understudy) score is a metric used to evaluate machine translation systems, where multiple good translations can exist for a given input. It compares the machine-generated translation to one or more human-generated reference translations. BLEU calculates a modified precision by counting n-gram matches (unigrams, bigrams, etc.) between the generated and reference translations, clipping counts to the maximum frequency in any reference. These precisions (P1, P2, P3, P4) are averaged, exponentiated, and adjusted by a brevity penalty (BP) to penalize overly short translations. While not perfect, the BLEU score provides a single real-number metric that has significantly aided machine translation development. It's also used in other text generation tasks like image captioning.

Attention Model

modification of attention,
already been using encoder - decoder pair,
the change:

- the remembering part is not been done properly for the long sequences, the connection between the encoder and decoder is too thin for such job.

- think how a human does it, processes it part by part, memorizing and processing at once is hard.

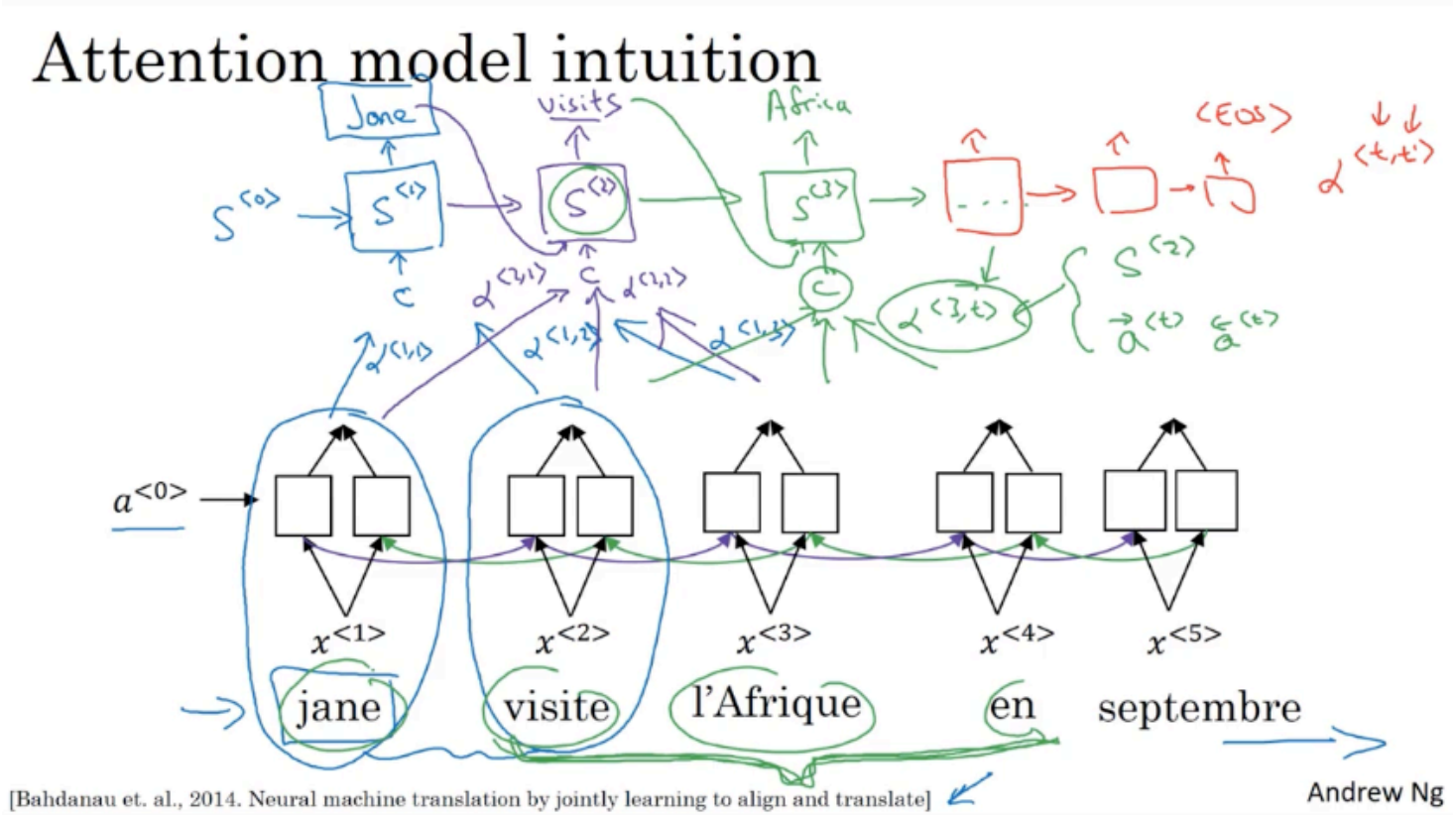


so it is an hard task for the model to remember long sequences

Attention Model Intuition

first developed for machine translation, later use of it generalized (a lot).

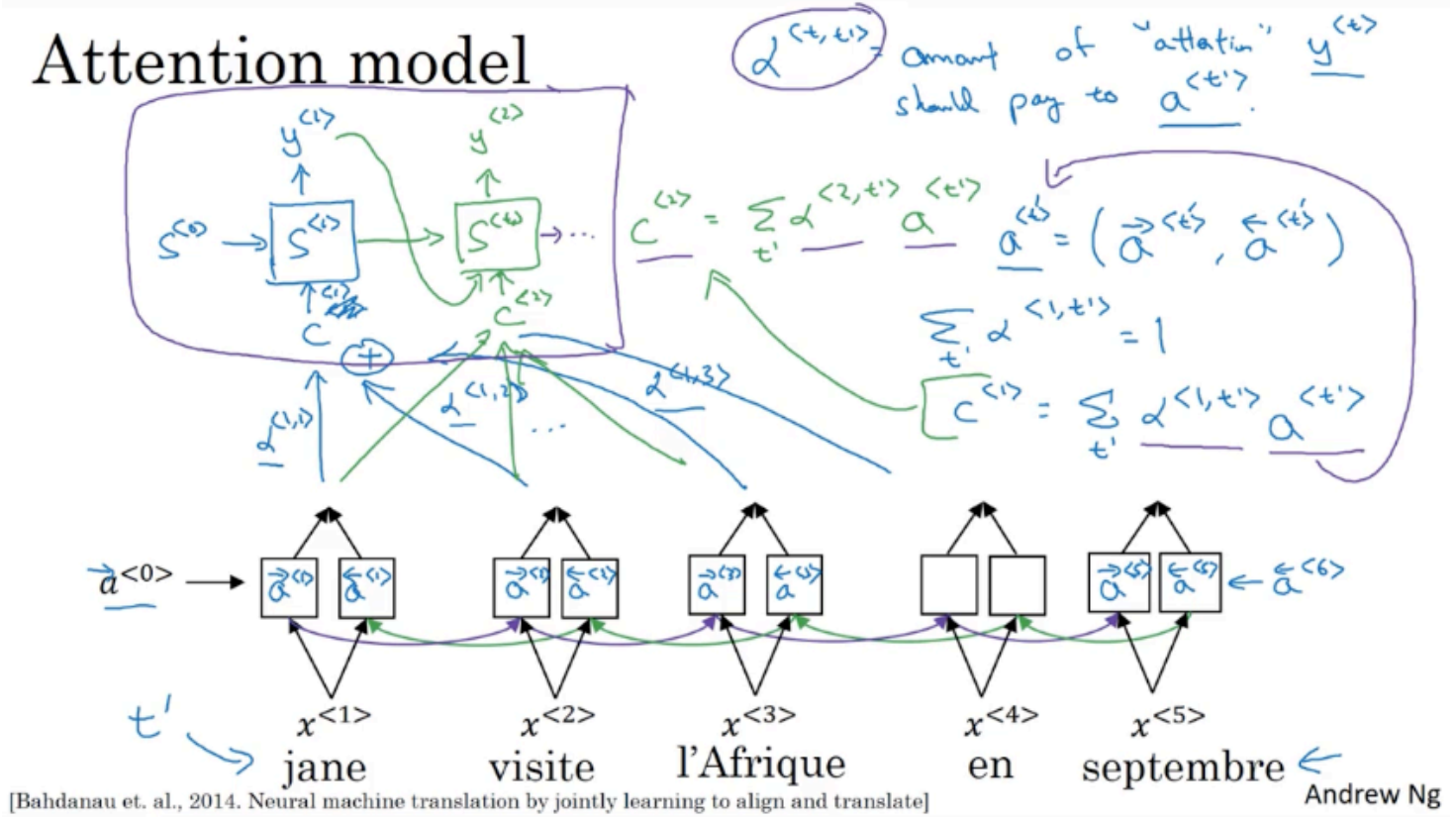
how much attention should we pay for each word in the sentence?



for t, how much attention is needed for t' (top right corner)

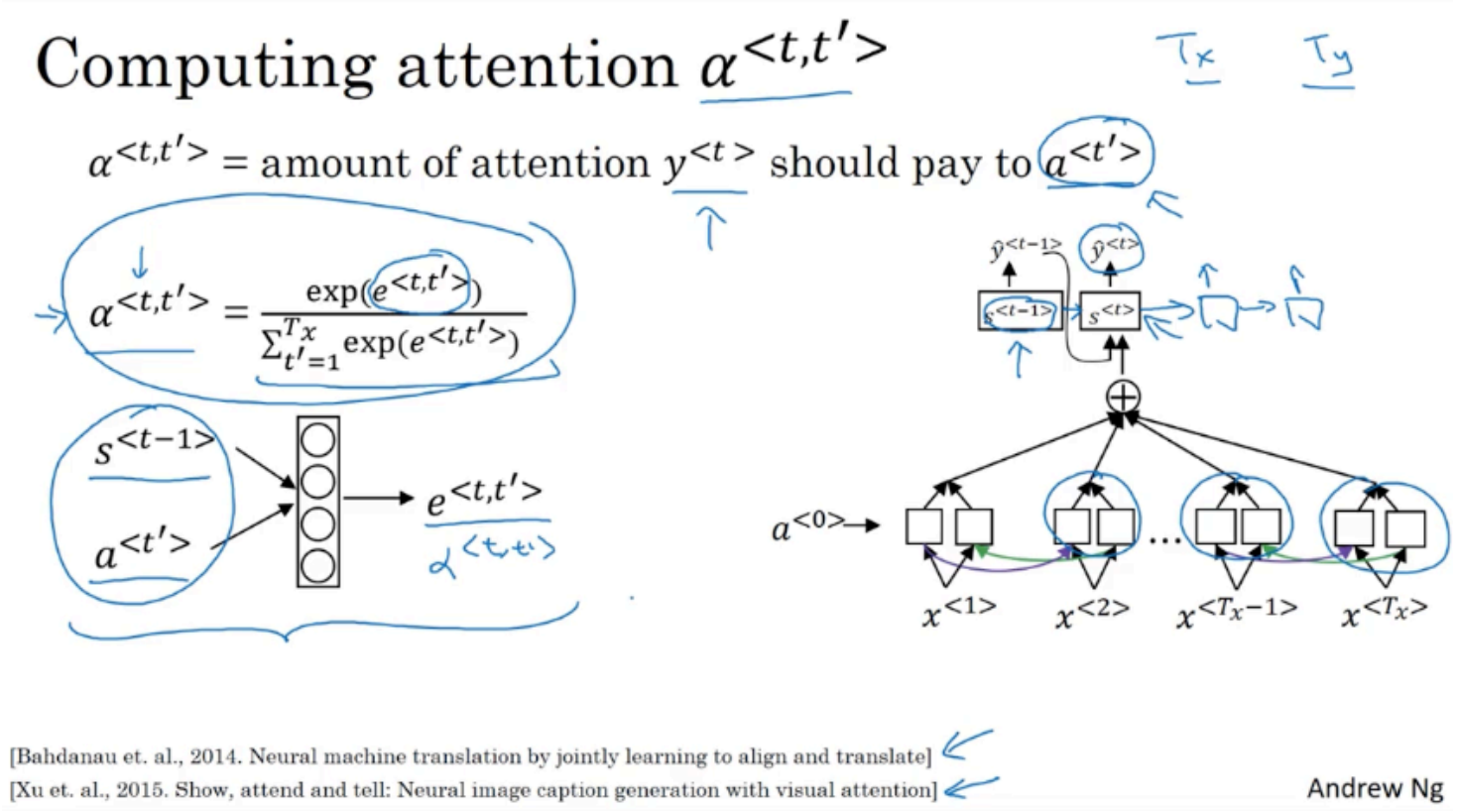
Attention Model

model pays attention pays attention to part of the input (more to some, less to some)



upper part seems like a regular RNN structure

how to define the attention weights (alpha value in purple):

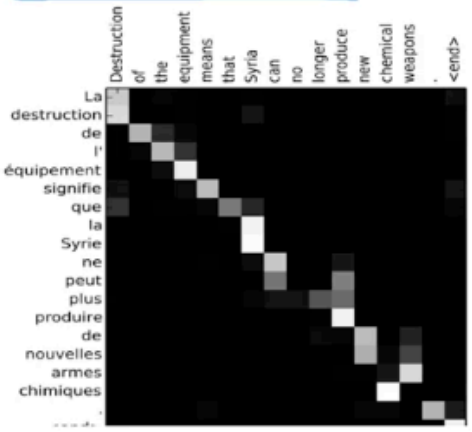


Attention examples

July 20th 1969 ⟶ 1969 – 07 – 20

23 April, 1564 ⟶ 1564 – 04 – 23

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



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

Andrew Ng

Speech Recognition

One of the best things about sequence to sequence modeling: highly accurate speech recognition

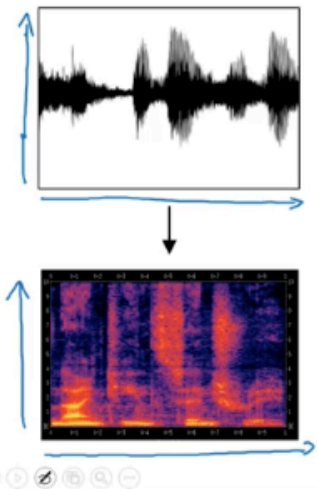
little changes in air pressure - ear, and microphone senses this

- common preprocessing step: process raw audio, generate a spectrogram. how loud is this audio for different frequencies. (human ear also does something that is similar to this)
- at first it was general to transcribe audio with phonemes. now it is not necessary. (or any other hand engineered representations)
- couple thousands -> many research papers are written on such amounts of data.
- couple hundred thousands -> commercial use.

Speech recognition problem

x ⟶ y

audio clip ⟶ transcript



"the quick brown fox"

→ phonemes: de kwik brawn

300h

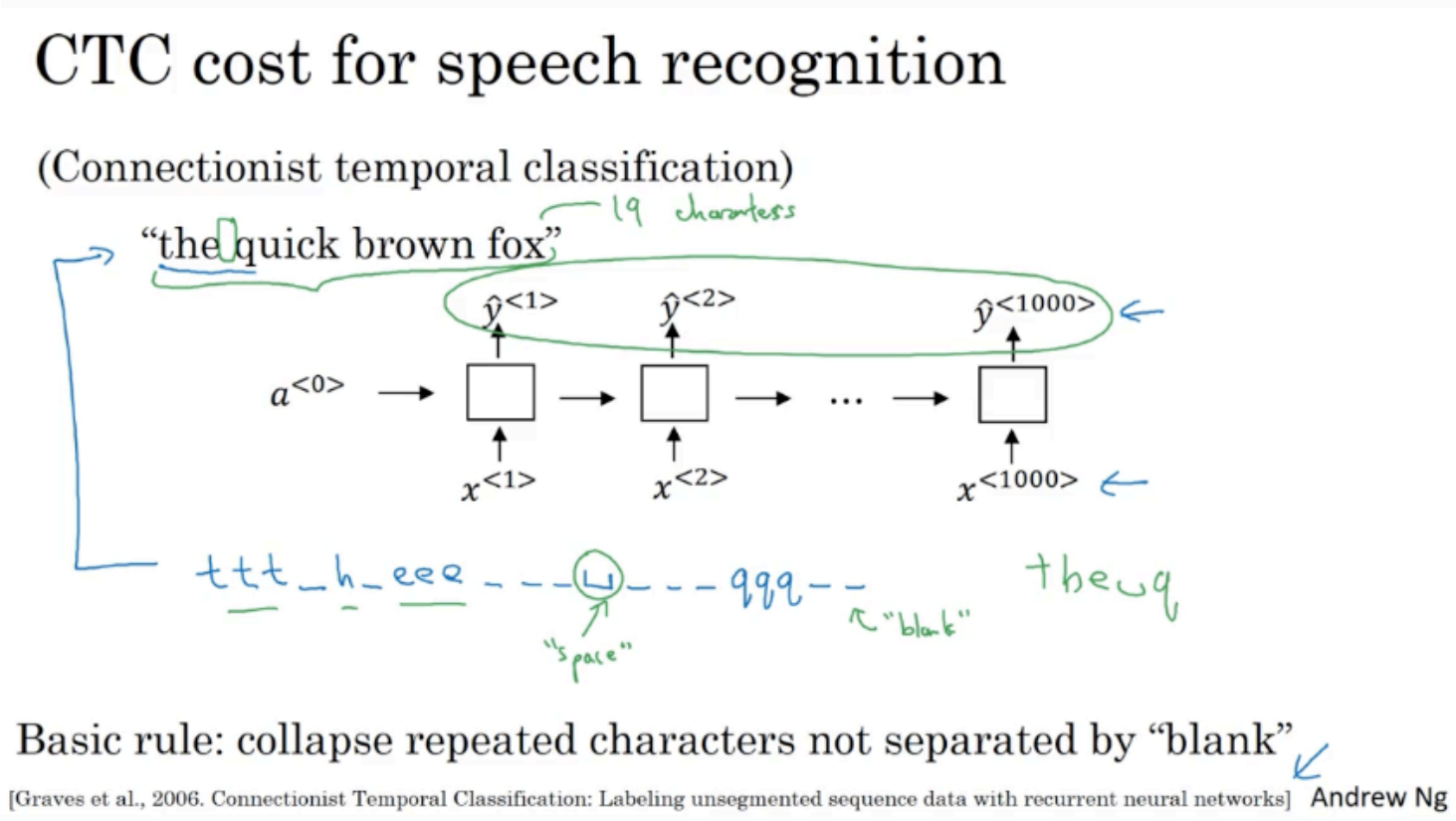
3000h

100,000h

Andrew Ng

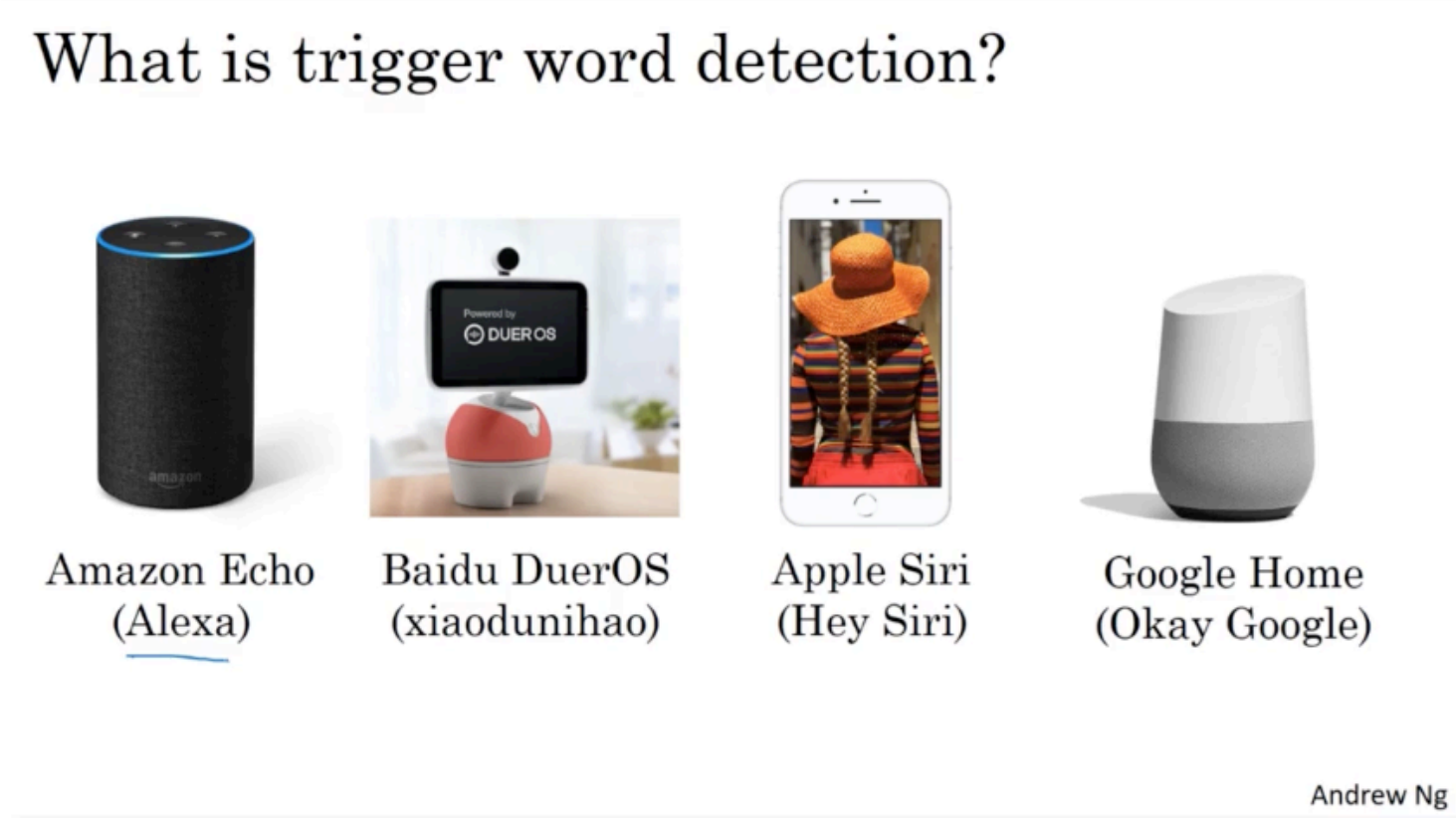
- you may use attention mechanism
- you may use CTC cost

generally # of time steps in the audio is much bigger than the output. (10 hz x 10 seconds, output is not even near it.)
soln:



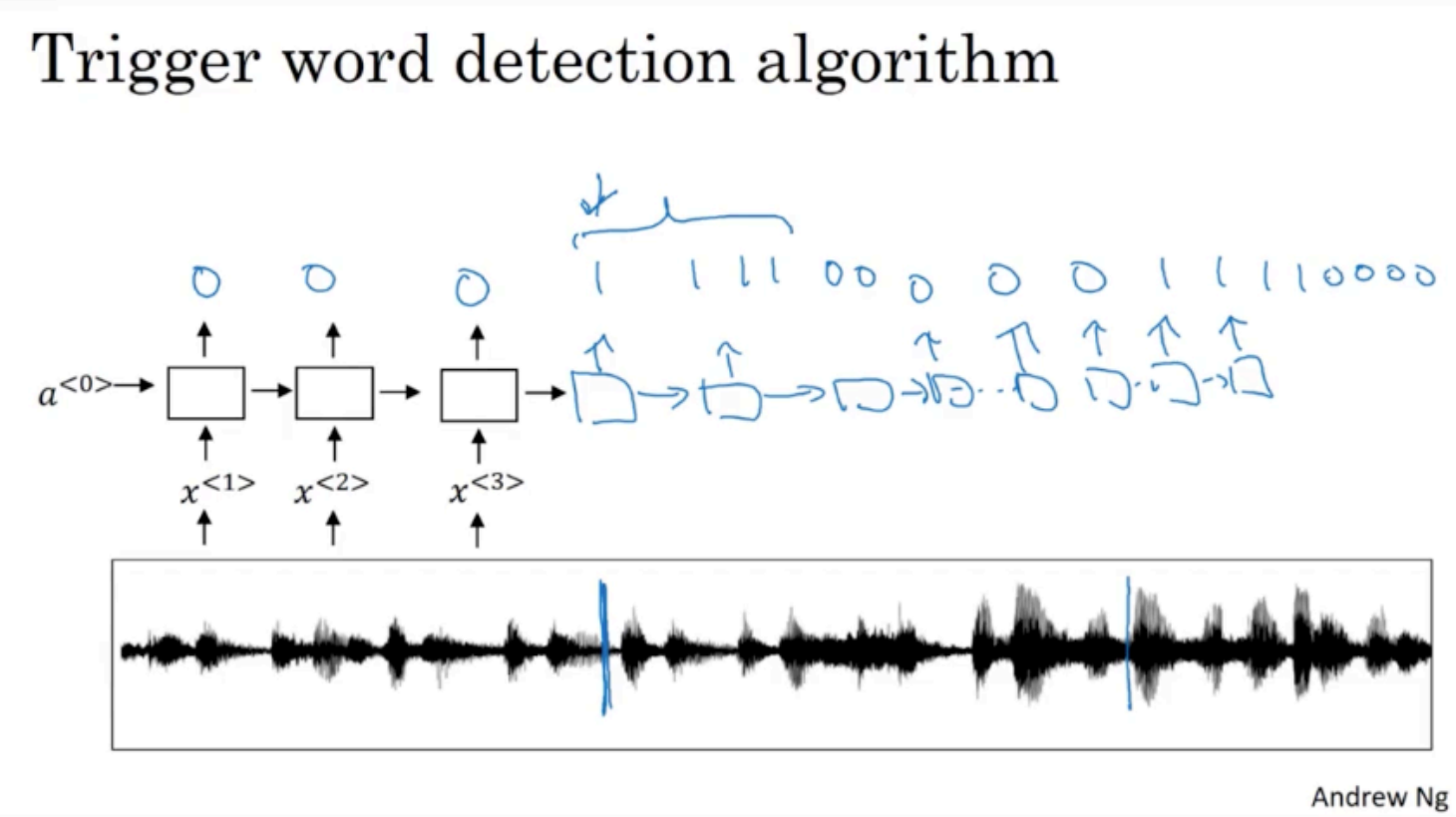
Trigger Word Detection

things you can wake up with your voice
Examples:



there is not a 'best algorithm' yet - may be old info.

Example algorithm:



instead of single 1, may put couple of 1's for trigger instances. - a little hack - Andrew Ng

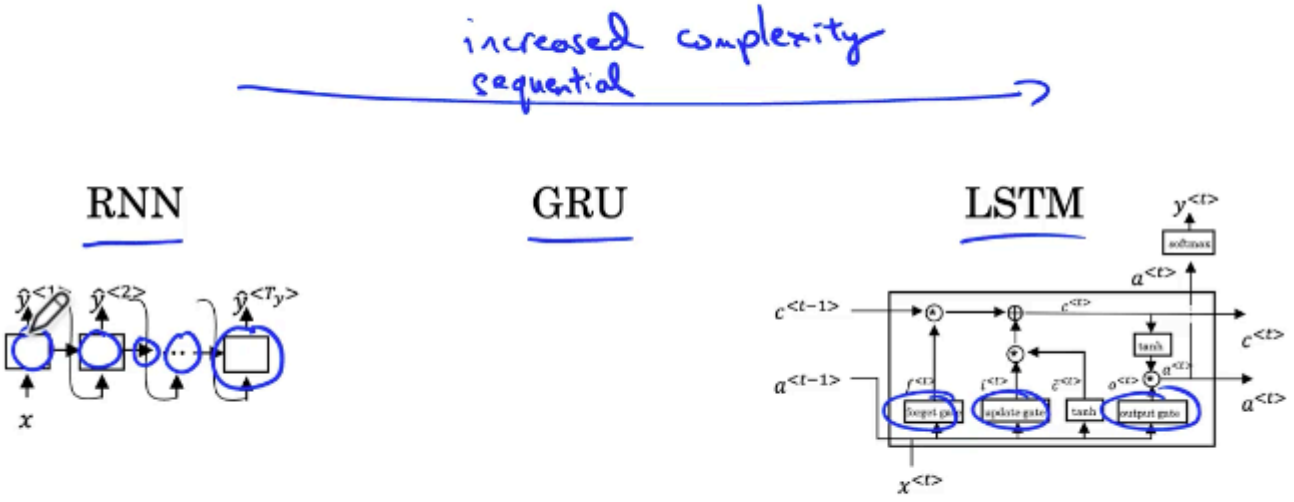
---

- Data synthesis is an effective way to create a large training set for speech problems, specifically trigger word detection.
- Using a spectrogram and optionally a 1D conv layer is a common pre-processing step prior to passing audio data to an RNN, GRU or LSTM.
- An end-to-end deep learning approach can be used to build a very effective trigger word detection system.

Transformer Network

Best models are often on transformer architecture
New and SOTA

Transformer Network Motivation

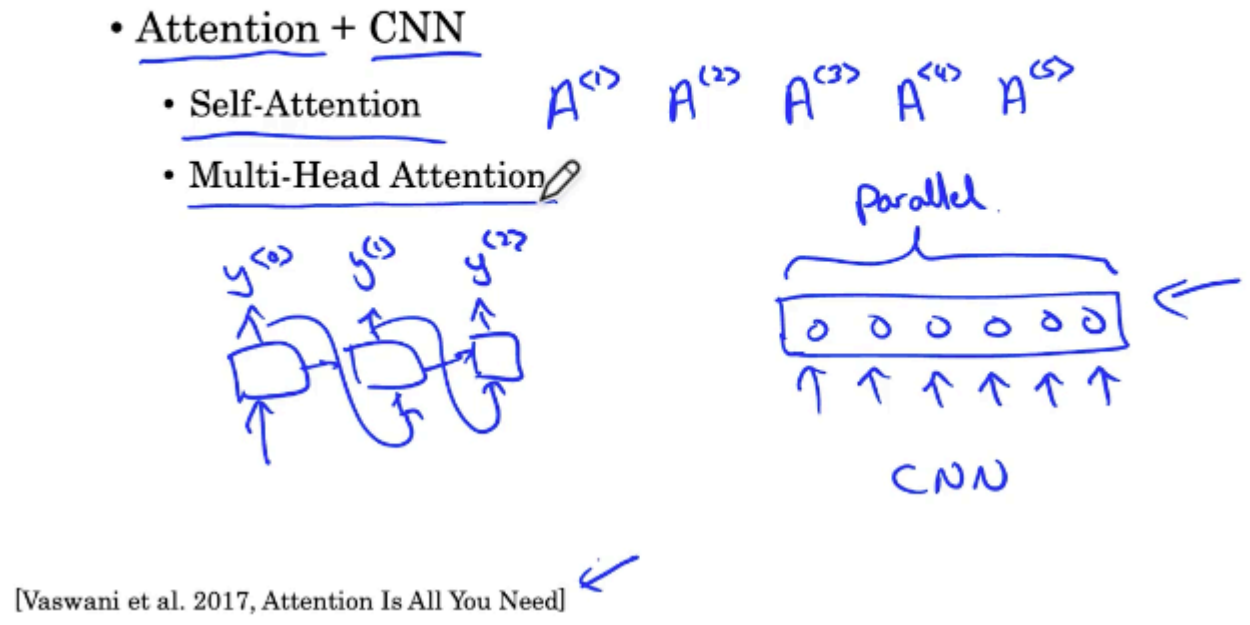


Andrew Ng

each unit is in essence a bottleneck by itself

- transformer solves it by enabling to process everything at once.

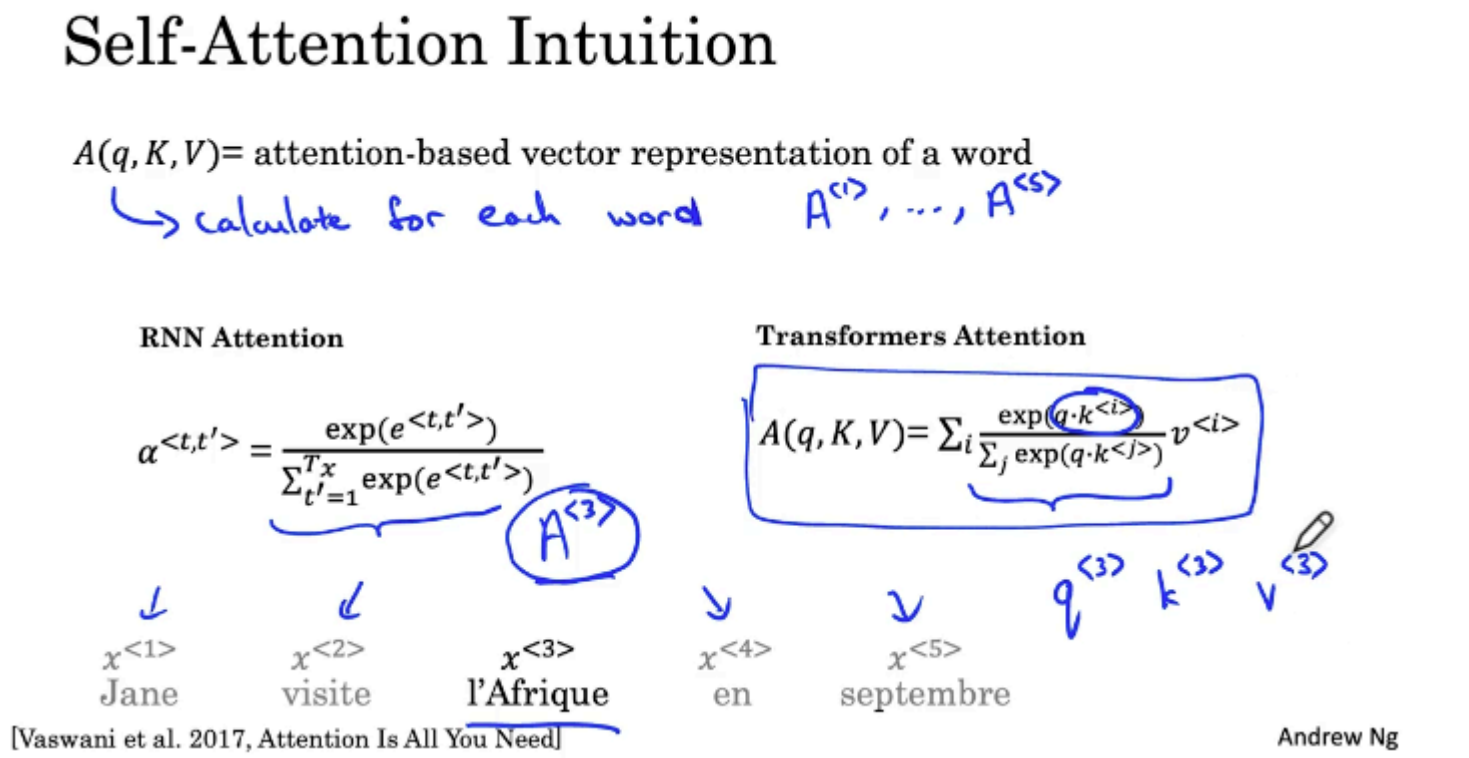
Transformer Network Intuition



[Vaswani et al. 2017, Attention Is All You Need]

Andrew Ng

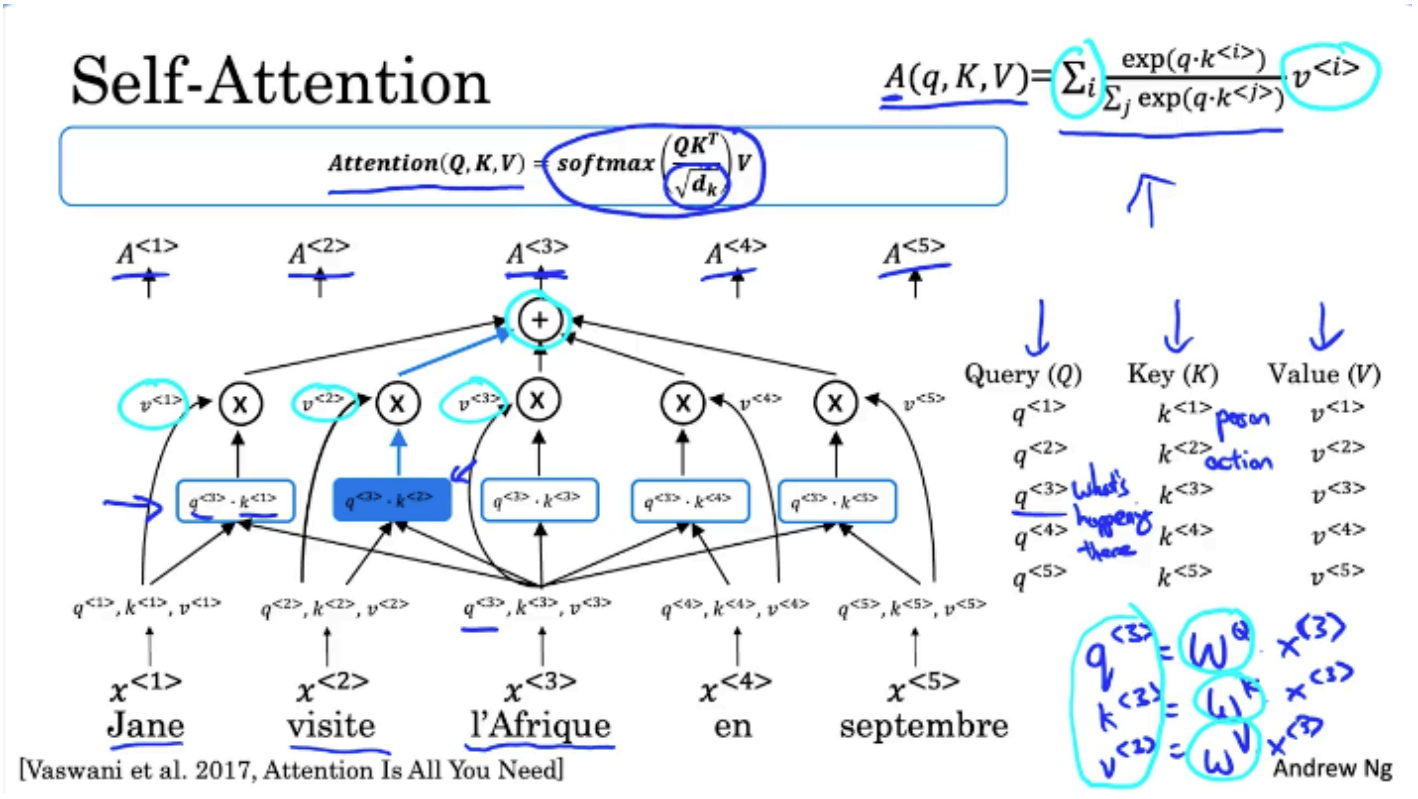
Self-Attention



how you use some word may affect how you want to represent it - continent, destination, place
both have softmax parts
main difference: for every word, have 3 values called query, key, value = attention value

Self Attention Representation A<3>

- q<3> question you get to ask - what's happening there?
- k<1> how good is the answer to this question - (this is repeated for each word in the sentence: k<2,3,4..>)
- maybe k<2> is most relevant: highest score (let's say)

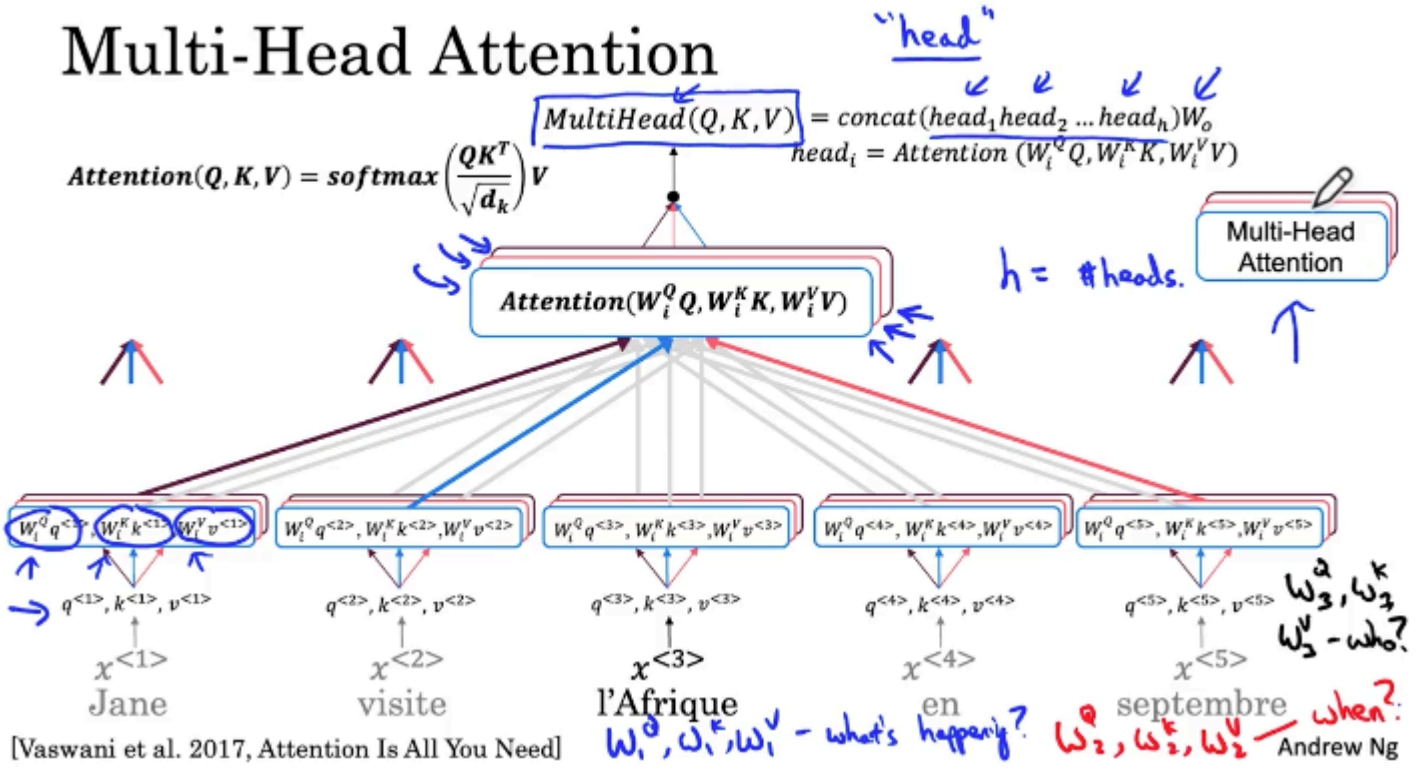


square root dk prevents it from exploding

Multi-Head Attention

A big for loop over the self-attention mechanism

- first question: what's happening?
- now there are other questions - other heads: when's happening? - happening to who? etc. (imaginary)

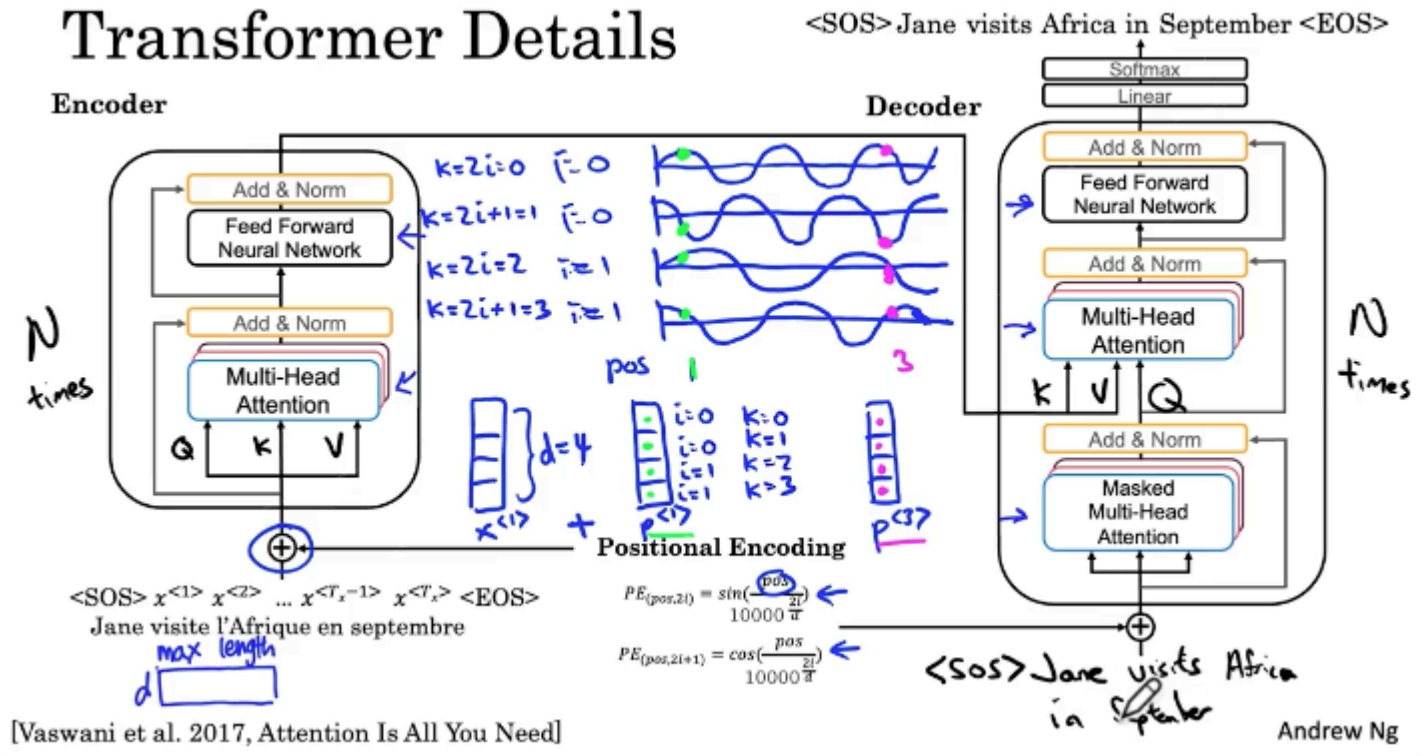


[Vaswani et al. 2017, Attention Is All You Need]

Multi-Head Attention

- **Concept:** Computes self-attention multiple times ("heads") in parallel, each with different learned weight matrices (W^Q, W^K, W^V). Each head, can be thought of answering a question, such as who, when, and what's happening.
- **Process:**
  1. For each head *i*:
     - Linearly project input Queries (Q), Keys (K), and Values (V) using learned matrices W_i^Q, W_i^K, W_i^V.
     - Compute self-attention: Attention(QW_i^Q, KW_i^K, VW_i^V)
  2. Concatenate the outputs of all heads.
  3. Multiply the concatenated output by a final weight matrix W^O.
- **Formula:**
  MultiHead(Q, K, V) = Concat($head_1$, $head_2$, ..., $head_h$)W^O
  where $head_i$ = Attention(QW_i^Q, KW_i^K, VW_i^V)1
- **Benefit:** Allows the model to attend to different aspects of the input sequence simultaneously (e.g., "what's happening," "when," "who"), capturing richer contextual information.
- The number of heads is represented by h.
- The computations of all heads can occur in parallel.
  The multi-head self-attention calculation is identical to standard self-attention; the difference is that it is performed multiple times with distinct, learned linear projections of Q, K, and V.

Transformer Network



Refer to the attached figure for the overall architecture of the transformer (from "Attention Is All You Need"), which includes both an **encoder** and a **decoder**. Below is a brief walkthrough of how it all fits together:

1. Encoder

• **Embeddings + Positional Encoding**: Each word/token is turned into an embedding and **positional encodings** (sine/cosine signals) are added to capture word order.

• **Multi-Head Self-Attention**: The encoder uses **Q**, **K**, **V** matrices (all derived from the embeddings) to allow each word to attend to other words. Having multiple heads enables the model to focus on different relationships in parallel.

• **Feed-Forward Network**: A fully connected layer processes the attention outputs.

• **Add & Norm**: Residual connections and a normalization layer (akin to batch normalization) help speed up learning and preserve positional information.

• The encoder block is typically repeated **N times** (e.g., 6).

2. Decoder

• The goal of the decoder is to produce the target sequence (e.g., an English translation).

• **Masked Multi-Head Attention**: The decoder first attends to its own past outputs (with future positions "masked" in training so the model can't "peek ahead").

• **Multi-Head Attention**: Next, the decoder attends to the **encoder's output** (queries from the decoder and K, V from the encoder).

• **Feed-Forward + Add & Norm**: Similar to the encoder, the attention output is passed through a feed-forward layer and normalization.

• This decoder block is also repeated **N times**, and finally feeds into a **linear + softmax** layer to predict the next token.

3. Training & Masking

• **Masking** is applied in the decoder's first attention layer to simulate predicting one word at a time, even though the entire correct sequence is known during training.
• This encourages the model to learn how to generate tokens step-by-step without cheating by looking ahead.

4. Takeaways

• The **encoder** captures contextual embeddings for the entire input sequence.
• The **decoder** uses its current output tokens (so far) to query the encoder's representation and generate the next token, step by step.
• **Positional encoding** ensures the model knows the relative position of words.
• **Residual connections** and **normalization** help stabilize and accelerate training.

Use these concepts (e.g., attention mechanisms, positional encodings, mask during training) as the basis for understanding transformer variants such as **BERT** and **DistilBERT**.

Q = interesting questions about the words in a sentence

K = qualities of words given a Q

V = specific representations of words given a Q

- The combination of self-attention and convolutional network layers allows of parallelization of training and *faster training*.
- Self-attention is calculated using the generated query Q, key K, and value V matrices.
- Adding positional encoding to word embeddings is an effective way to include sequence information in self-attention calculations.
- Multi-head attention can help detect multiple features in your sentence.
- Masking stops the model from 'looking ahead' during training, or weighting zeroes too much when processing cropped sentences.

References

Week 1:

- Minimal character-level language model with a Vanilla Recurrent Neural Network, in Python/numpy (GitHub: karpathy)
- The Unreasonable Effectiveness of Recurrent Neural Networks (Andrej Karpathy blog, 2015)
- deepjazz (GitHub: jisungk)
- Learning Jazz Grammars (Gillick, Tang & Keller, 2010)
- A Grammatical Approach to Automatic Improvisation (Keller & Morrison, 2007)
- Surprising Harmonies (Pachet, 1999)

Week 2:

- Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings (Bolukbasi, Chang, Zou, Saligrama & Kalai, 2016)
- GloVe: Global Vectors for Word Representation (Pennington, Socher & Manning, 2014)
- Woebot.

Week 4:

- Natural Language Processing Specialization (by DeepLearning.AI)
- Attention Is All You Need (Vaswani, Shazeer, Parmar, Uszkoreit, Jones, Gomez, Kaiser & Polosukhin, 2017)