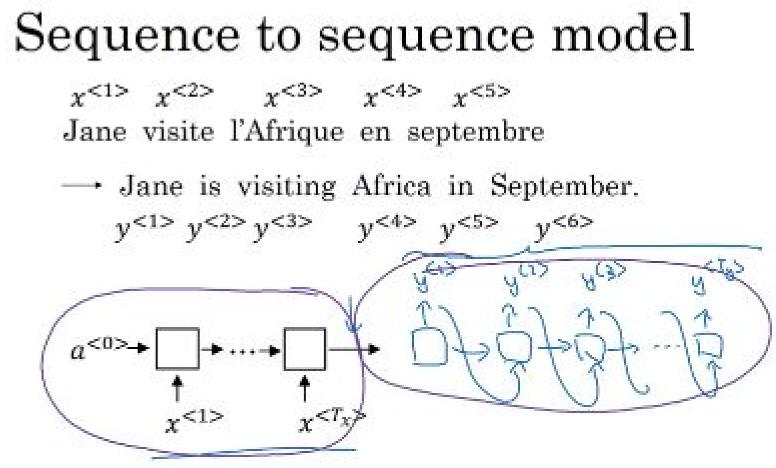
Andrew Ng's Recurrent Neural Networks

Week 3: Sequence Models & Attention Mechanisms

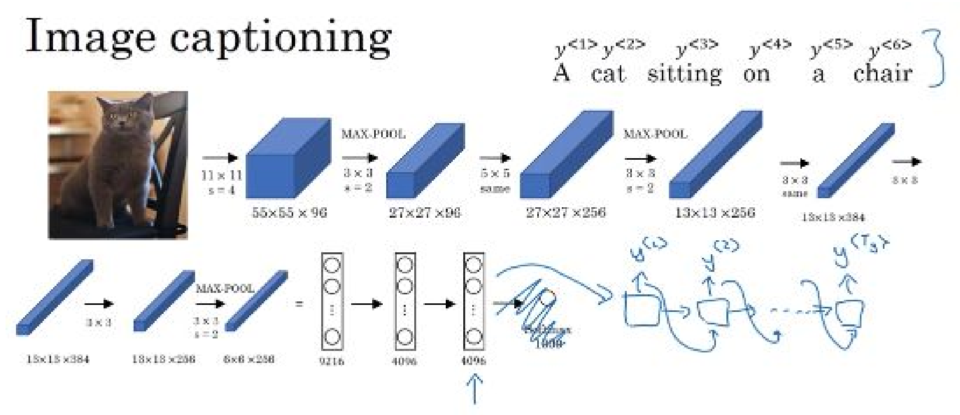
Basic Models

- This week we are going to hear about sequence-to-sequence models. Let's start with basic models. Suppose we have a french sentence that need to be translated to english. The french words will be denoted with $x^{<1>}$, $x^{<2>}$... and the output in english will be denoted by $y^{<1>}$, $y^{<2>}$, ... First we are going to have an encoder network (in purple below) whose job is to output a vector that represents the input sentence. We then have a decoder network which will be trained to output the french translation one word at a time.



Sequence to sequence model
$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$
Jane visite l'Afrique en septembre
→ Jane is visiting Africa in September.
$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$
$a^{<0>}$
$x^{<1>}$
$x^{<T_x>}$
$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<T_y>}$

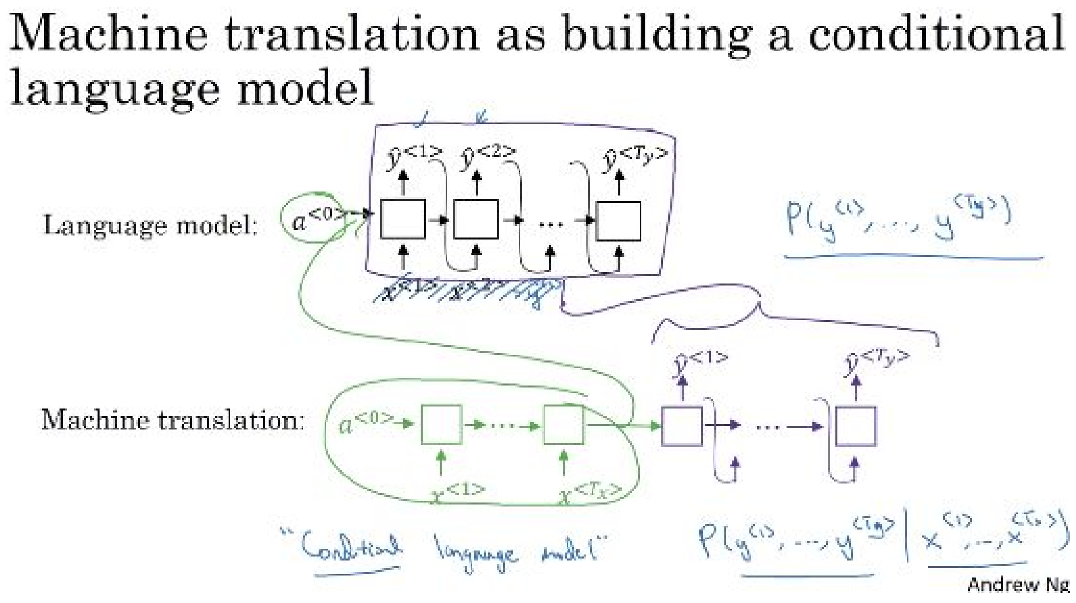
- This type of architecture is also useful in other situations. The example below is for generating a captioning sentence for an image.



In this case the encoder network is the convolutional neural network (an Alexnet in this case) without the last softmax activation. So we end up with a 4096-dimensional vector that represents the image. This vector is then fed as input to the decoder network, an RNN, whose job is to come up with a captioning sentence.

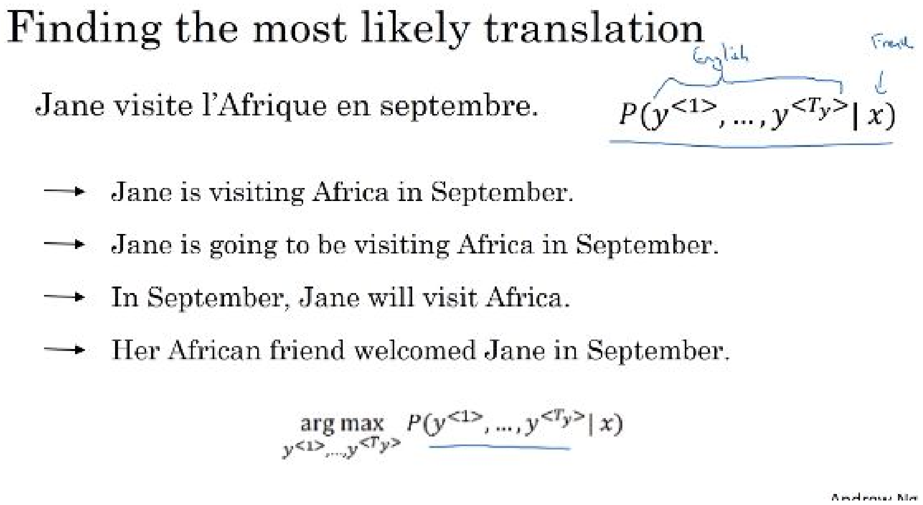
Picking the most likely sentence

- There are some similarities between a language model and a sequence to sequence machine translation model. The first model below is the one we learned in the first week of the class. The second one is the machine translation one we built in the last session. Notice how the decoder part looks just like the language model from week 1. The difference is the machine learning model is a conditional learning model because it expects a sentence as an input so its probability is expressed as $P(y^{<1>}, y^{<2>}, ..., y^{<t_y>}|x^{<1>}, ..., x^{<t_n>})$. So we are trying to estimate the probability of an English translation given as input a french sentence.



Machine translation as building a conditional language model
Language model:
$a^{<0>}$
$\hat{y}^{<1>}$ $\hat{y}^{<2>}$ $\hat{y}^{<T_y>}$
$x^{<1>}$ $x^{<2>}$
$P(y^{<1>}, \dots, y^{<T_y>})$
Machine translation:
$a^{<0>}$
$x^{<1>}$ $x^{<T_x>}$
$\hat{y}^{<1>}$ $\hat{y}^{<T_y>}$
"Conditional language model"
$P(y^{<1>}, \dots, y^{<T_y>} | x^{<1>}, \dots, x^{<T_x>})$
Andrew Ng

- Given this, we can then apply the model to translate from french to english and it will tell us the probabilities of different english translations of the input sentence. These translations we get from a distribution that we do not sample at random. Instead we are trying to find the english sentence that maximizes that probability. So we need to find an algorithm that helps us do that. The most common one is called beam search and we will look at it in the next session .



Finding the most likely translation

Jane visite l'Afrique en septembre. $\underline{P(\overbrace{y^{<1>}, \dots, y^{<T_y>}}^{\text{English}} | \overset{\text{French}}{x})}$

→ Jane is visiting Africa in September.

→ Jane is going to be visiting Africa in September.

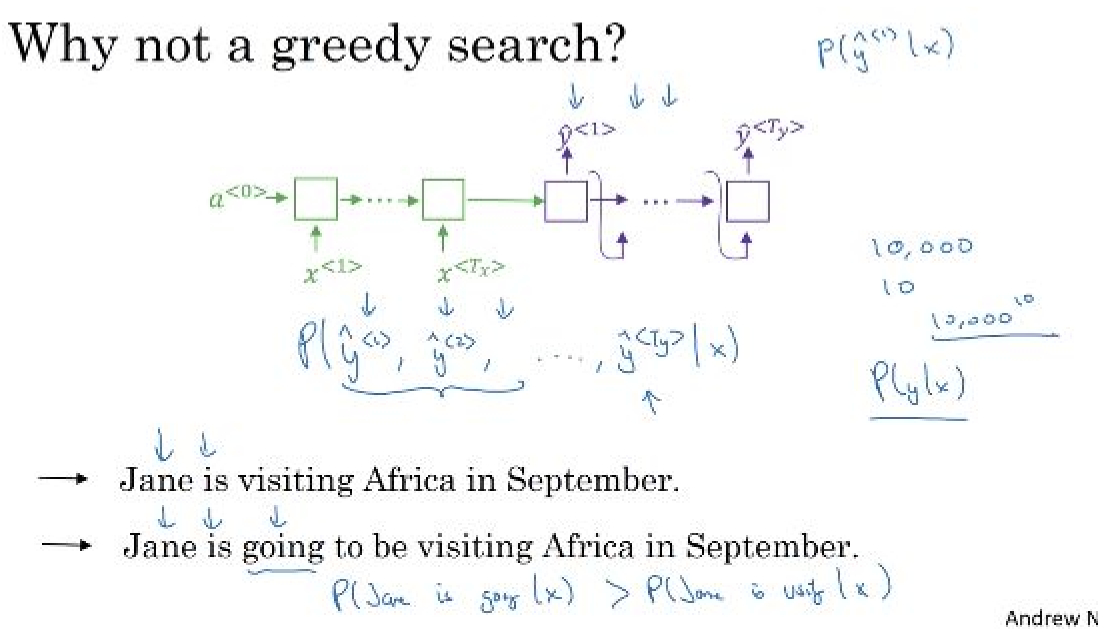
→ In September, Jane will visit Africa.

→ Her African friend welcomed Jane in September.

$$\underset{y^{<1>},\dots,y^{<T_y>}}{\arg\max} P(\underline{y^{<1>}, \dots, y^{<T_y>}} | x)$$

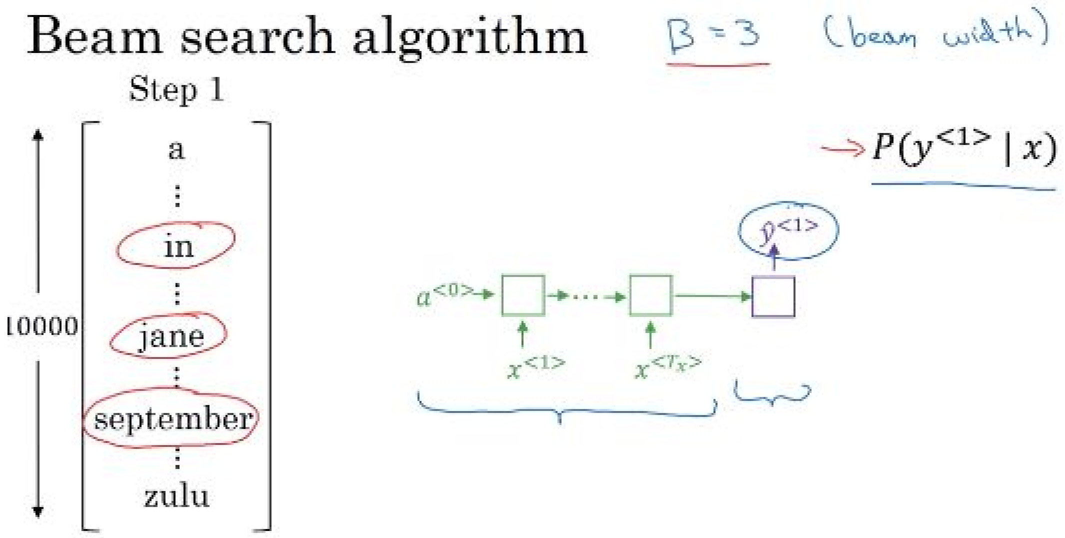
Andrew Ng

- Why not use greedy search? Well, it turns out it doesn't work very well. Greedy search picks the most likely word, one word at a time and that approach will not yield the best translation. Consider the 2 translations below. The first translation is better but it could be that based just on the first 3 words P(Joe is going | x) > P(Joe is visiting | x) and this will cause us to pick the 2nd translation. The number of word combinations in the english language is very big and we cannot possibly rate them all so we need to use an approximate search algorithm which will try to pick the sentence that maximizes the conditional probability. Usually, it does a good enough job.

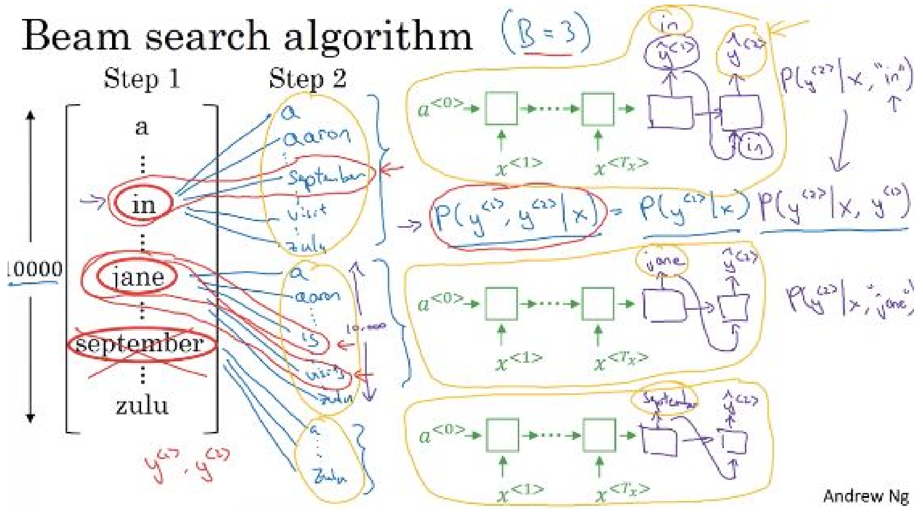

Why not a greedy search?
$P(\hat{y}^{<1>}|x)$
$a^{<0>}$
$x^{<1>}$
$x^{<T_x>}$
$\hat{y}^{<1>}$
$\hat{y}^{<T_y>}$
$P(\hat{y}^{<1>}, \hat{y}^{<2>}, \ldots, \hat{y}^{<T_y>}|x)$
10,000
10
$10{,}000^{10}$
$P(y|x)$
Jane is visiting Africa in September.
Jane is going to be visiting Africa in September.
P(Jane is going | x) > P(Jane is visiting | x)
Andrew N

Beam Search

- Beam search is the most widely used algorithm to pick the most likely translation. Let's see how it works. In step 1 Beam search has to pick the most likely english word so I have a vocab of size 10,000 on the left and we have an encoder network in green below. Once we have the output of the encoding step we will try to pick the best word. Whereas greedy search would only pick the first word and then move on, beam search can consider multiple alternatives. Beam search uses a parameter b which is the beam width which is set = 3 in our example below. What this means is that beam search will consider 3 possibilities at a time for the first word and will keep them in memory. In our example below, the 3 possibilities are in, Jane & september. To be clear, we run the french sentence through the encoder network and the output vector is given to the decoder network. In the first step of the decoder network we have a softmax activation with 10,000 possibilities and we keep in memory the top 3.



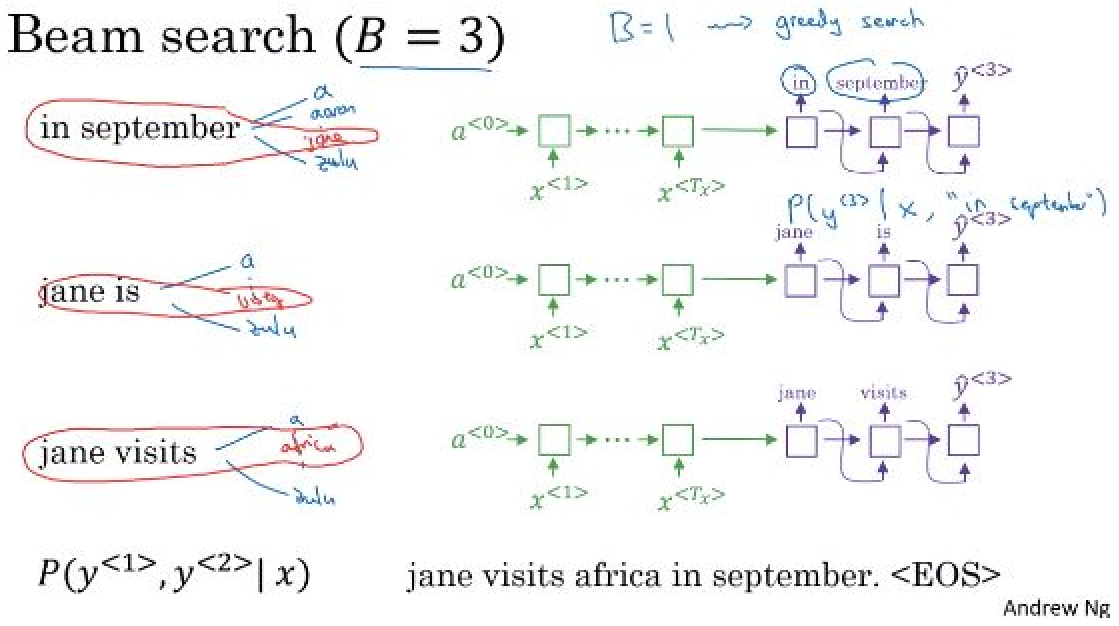
- Now we move on to the 2nd step of beach search.



What we do in this step is to consider the 2nd word given the 3 possibilities we have for the 1st word. For each of the 3 possibilities what happens is that the first word, in for example is fed to the 2nd layer of the decoder network and that 2nd layer is computing the probability $P(y^{<2>}|x, \text{'in'})$ and we do the same for the other 2 probabilities (jane and september. Now we can compute the probability of the first 2 words as follows: $P(y^{<1>}, y^{<2>}| x) = P(y^{<1>}| x) \ * \ P(y^{<2>}| x, y^{<2>})$. Notice a couple of things: first that we are making 3 copies of our network with different choices for the first word so we have 30,000 choices to evaluate for the 2nd word.

- Let's look at one more step of beam search. In step number 3 we have 3 probabilities (because B=3) for the first 2 words and we are now going to look for the 3rd word. We do the same as in the previous step. Each choice has a copy of the 10,000 vocab and will eventually select one choice. It does it using our decoding network where our already selected $y^{<1>}$ is passed to the 2nd layer and our already selected $y^{<2>}$ is passed to the 3rd layer. We can then express the probability of $y^{<3>}$ for our first choice as

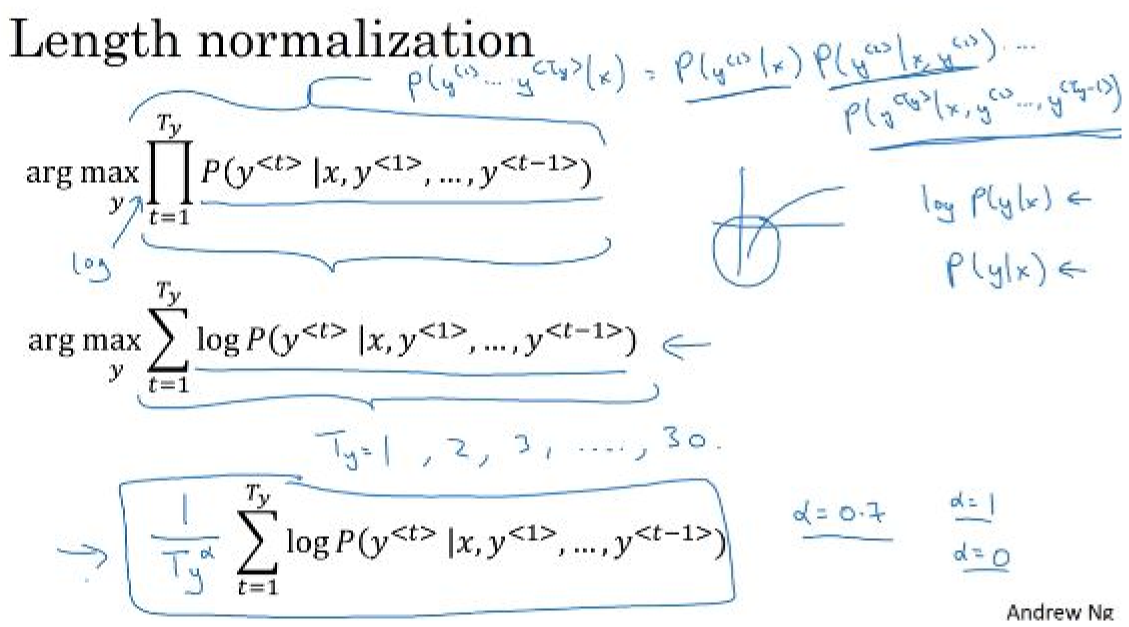
$P(y^{<3>} | x, \text{'in september'})$. We do the same step for the 2 other choices we have. We have 3 possibilities because B=3, notice that if B=1 then beam search becomes greedy search.



This is how beam search works. In the next session we'll see some tips and tricks to make beam search work better.

Refinements to Beam Search

- Let's look at a few enhancements go beam search. The first one is length normalization. Let's see what it does. Beam search is trying to maximize the probability shown below. All these probabilities are < 1 and usually small so multiplying them by each other will get us close to zero and sometimes will cause numerical underflow. In practice, instead of maximizing the product below we do logs. Our new probability is then show in the middle equation below. By taking logs we end up with a more stable algorithm that is less prone to rounding errors or numerical underflow.



Length normalization

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

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

log

$\log P(y|x) \leftarrow$

$P(y|x) \leftarrow$

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

$T_y = 1, 2, 3, \dots, 30.$

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

$\alpha = 0.7$

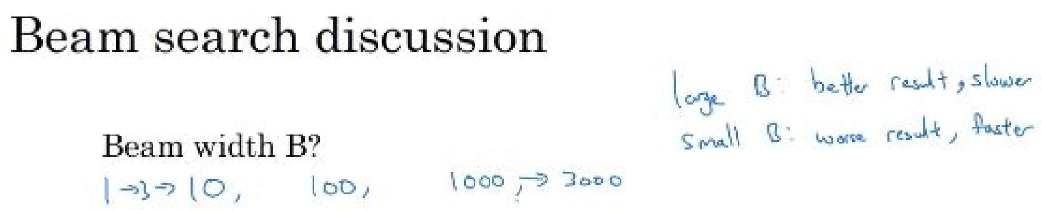
$\alpha = 1$

$\alpha = 0$

Andrew Ng

Another thing we do is add a normalization term, $1/T_y^{\alpha}$, so this takes the average of the log of the probability of each word and reduces the penalty for outputting longer translations.

- Our next discussion is how we pick the beam width B? We are using a B=3 in this course but in practice it is common to see widths from 10 to 100 to 1,000 to 3,000. Notice we will have diminishing returns as we get to the top part of the spectrum. If we have a large B we will get better results but it will be slower since we have more copies of the network and the process is more computationally expensive. If we have a small B we will have a worse result but it will run faster.



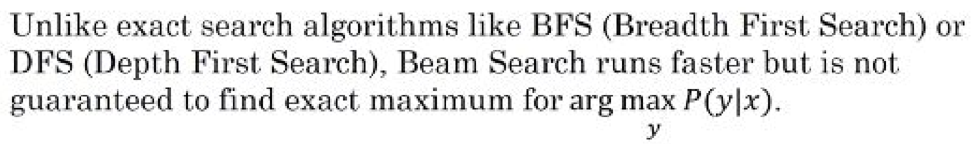
Beam search discussion

large B: better result, slower

small B: worse result, faster

Beam width B?

$1 \rightarrow 3 \rightarrow 10, \quad 100, \quad 1000 \rightarrow 3000$

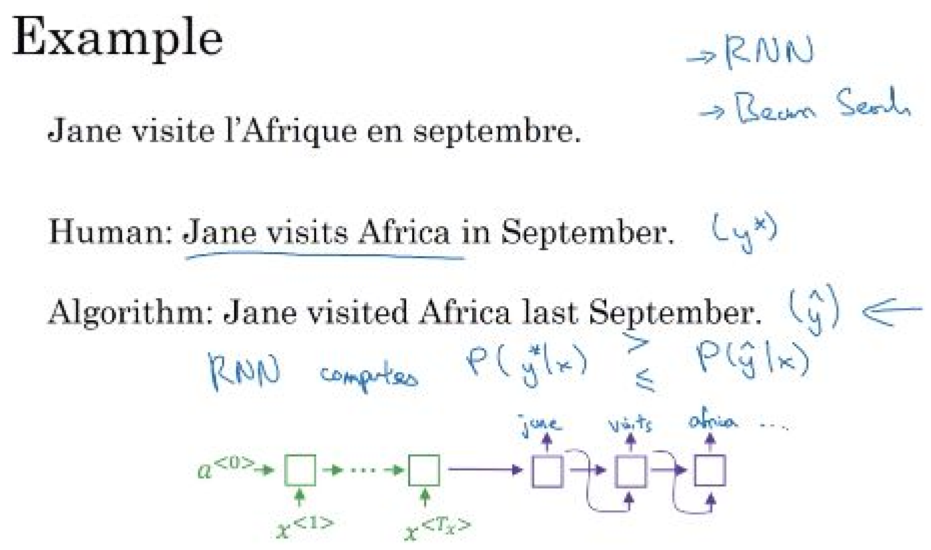


Unlike exact search algorithms like BFS (Breadth First Search) or DFS (Depth First Search), Beam Search runs faster but is not guaranteed to find exact maximum for $\arg\max_y P(y|x)$.

Finally, notice that unlike other algorithms like BFS or DFS, Beam search will run faster but is not guaranteed to find the exact maximum for the probability we are trying to maximize. BFS and DFS will find it.

Error Analysis in Beam Search

- In this session we are going to see how to perform error analysis in Beam search. Assume the first sentence below is the input X to our RNN and a human gave us a pretty good translation that we are going to call y^*. Let's also assume our model output is the 3rd sentence so it is $yhat$. The second translation is not very good and it would be nice to know if the problem is in our RNN or with Beam search. So what we do is have the RNN compute 2 probabilities: $P(y^*|x)$ and $P(yhat\,|\,x)$ and we compare the difference between the 2.



Example

Jane visite l'Afrique en septembre.

$\rightarrow$ RNN

$\rightarrow$ Beam Search

$\boxed{\text{B}\uparrow}$

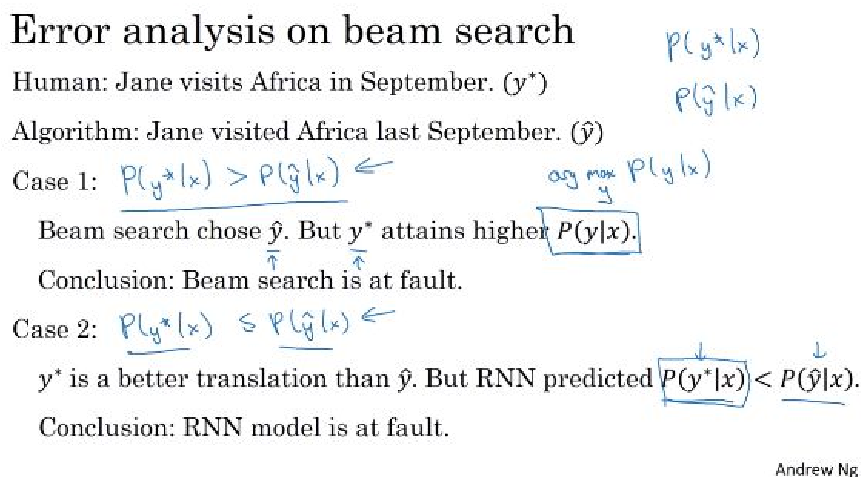
Human: Jane visits Africa in September. (y^*)

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

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

$a^{<0>}$
$x^{<1>}$
$x^{<T_x>}$
jane
visits
africa
...

- So what we do when we compare both probabilities. We have two cases:
  - *If* $P(y^*|x) > P(yhat\,|\,x)$ *then* the problem is Beam search because beam search's only job is to find the translation that maximizes the probability
  - *If* $P(yhat|\,x) > P(y^*\,|\,x)$ *then* we can conclude that the RNN is at fault because $yhat$ is a better translation but the RNN predicted y^*



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: $\underline{P(y^*|x) > P(\hat{y}|x)}$ $\leftarrow$ $\quad \arg\max_y P(y|x)$

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

Conclusion: Beam search is at fault.

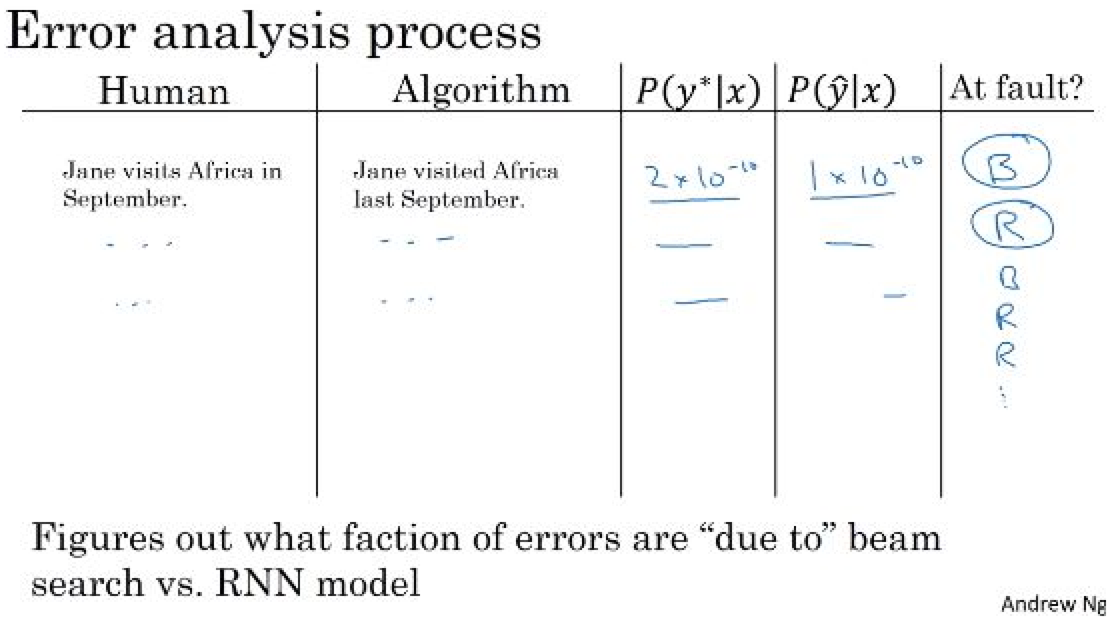
Case 2: $P(y^*|x) \le P(\hat{y}|x)$ $\leftarrow$

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

Conclusion: RNN model is at fault.

Andrew Ng

- We can then carry out error analysis to figure out what faction of errors are due to beam search and what are due to the RNN. Depending on who's at fault we can try to increase the parameter B if there are many errors due to beam search or we could try to do more analysis on the RNN to decide we should get more training data, add regularization or try a different network architecture.



Error analysis process

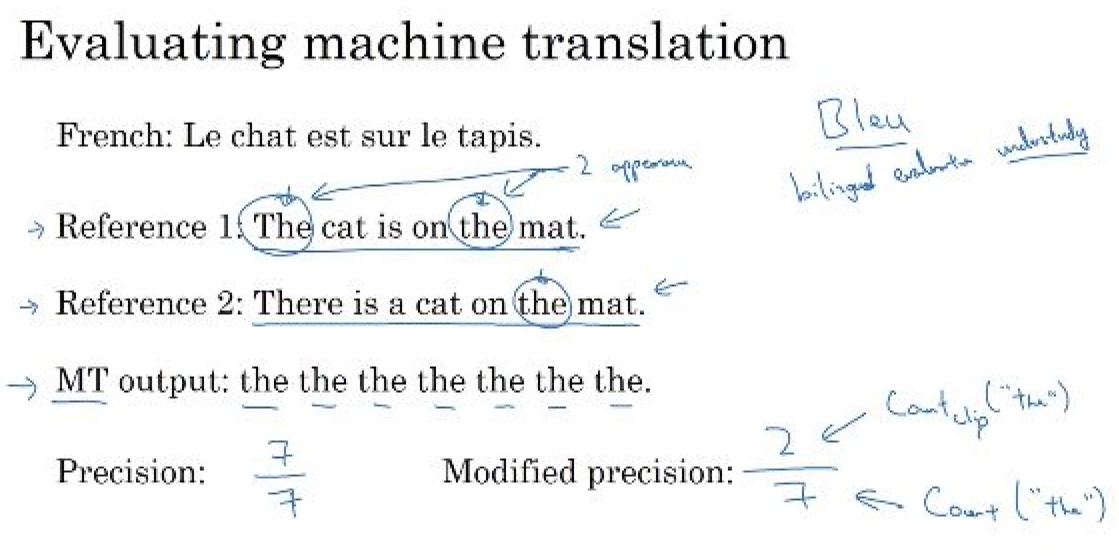
Human	Algorithm	$P(y^*\|x)$	$P(\hat{y}\|x)$	At fault?
Jane visits Africa in September.	Jane visited Africa last September.	2×10^{-10}	1×10^{-10}	B
- - -	- - -	—	—	R
...	...	—	—	B
				R
				R
				⋮

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

Andrew Ng

Bleu Score

- One of the challenges of Machine translation is that we can have many translations so the question becomes how do we measure their accuracy? We use something called the bleu score. Bleu stands for Bilingual Evaluation Understudy. Let's see how it works. We assume we have a french sentence to translate and two reference translations provided by humans. Finally, we have a machine translation output that is pretty bad (shown below). First we try to find out the precision. Precision is computed as dividing the number of times a word provided by machine translation appears in the references (7 in the case below because 'the' appears in both references and the whole translation is composed of 'the's') divided by the number of length of the machine translation output (7 in the case below). So this is a misleading indicator because the precision is high but we know that the translation is bad. What we do instead is we take a modified precision metric where we give credit to word only up to the maximum number of times it appears in the references. Modified precision is the 2/7 in this case since the max number of times 'the' appears in a reference sentence is 2.



Evaluating machine translation

Bleu
bilingual evaluation understudy

French: Le chat est sur le tapis.

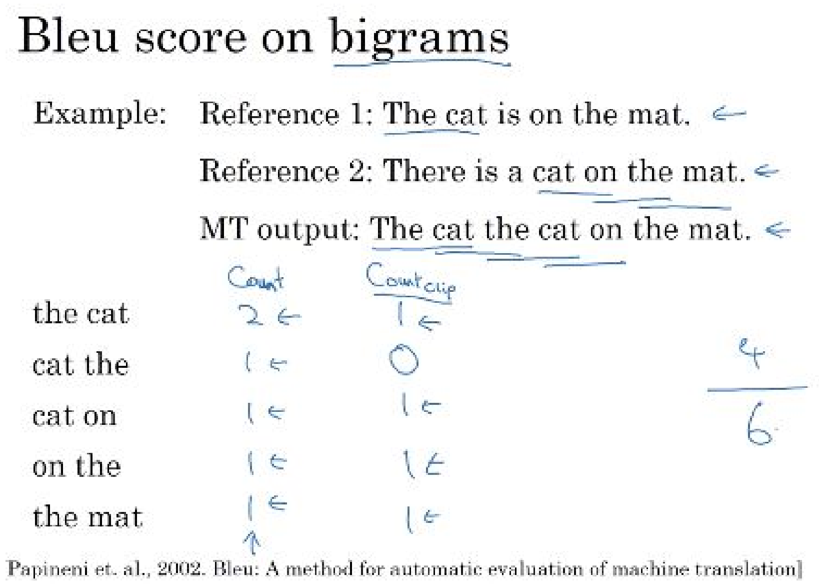
→ Reference 1: The cat is on the mat. ← (2 appearances)

→ Reference 2: There is a cat on the mat. ←

→ MT output: the the the the the the the.

Precision: $\frac{7}{7}$ Modified precision: $\frac{2}{7}$ ← Count_clip("the") / ← Count("the")

- So far we have been looking at words in isolation. We can also look at pairs of words or bigrams. Consider the example below. We have two human references and one machine translation output.



Bleu score on bigrams

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. ←

	Count	Count_clip
the cat	2 ←	1 ←
cat the	1 ←	0
cat on	1 ←	1 ←
on the	1 ←	1 ←
the mat	1 ←	1 ←

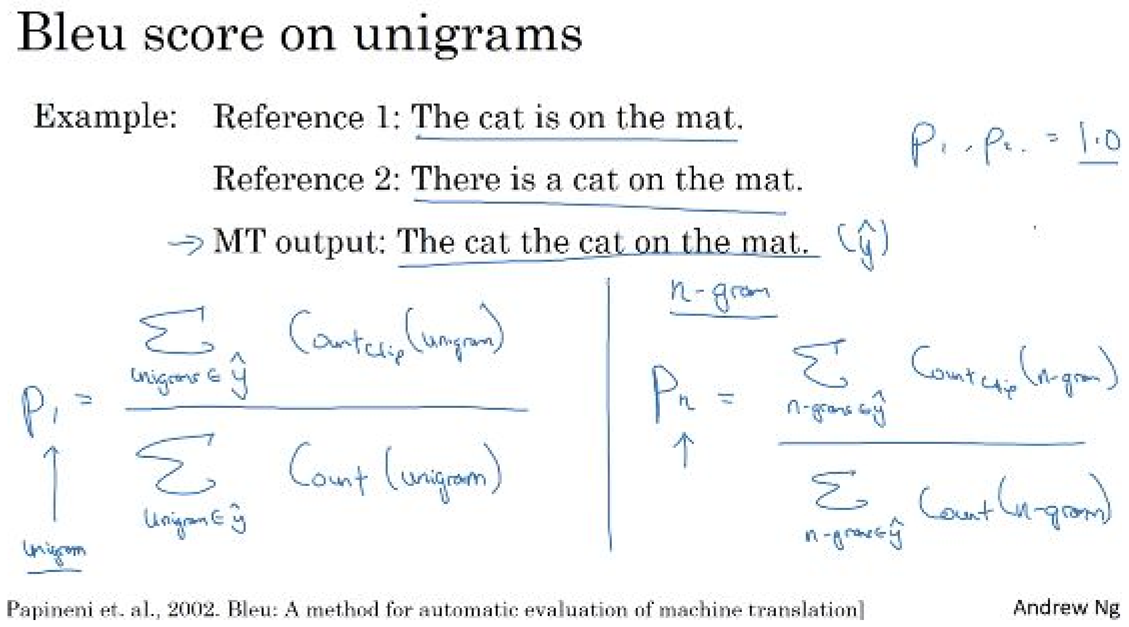
$\frac{4}{6}$

Papineni et. al., 2002. Bleu: A method for automatic evaluation of machine translation]

What we do next is we identify and count the bigrams that appear in the machine translation. We have 5 bigrams with one appearing twice (the cat). We then do the count clip to give the bigrams credit for the times they appear in our references: the cat appears once, cat the appears 0 and so on. We then end up with a modified precision of 4/6 where 4 is the number of times the bigrams appear in the references and 6 is the total number of bigrams.

- Let’s formalize this a little bit more. We’ll come up with the equations for the unigram case and for the n-gram case. For unigrams we have

$P_1 = \sum_{unigram\ \varepsilon\ yhat} Count_{clip}(unigram) / \sum_{unigram\ \varepsilon\ yhat} Count(unigram)$. For the n-gram case the equation is $P_n = \sum_{n-gram\ \varepsilon\ yhat} Count_{clip}(n-gram) / \sum_{n-gram\ \varepsilon\ yhat} Count(n-gram)$



Bleu score on unigrams

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. $(\hat{y})$

$P_1, P_2 = 1.0$

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

$\uparrow$ unigram

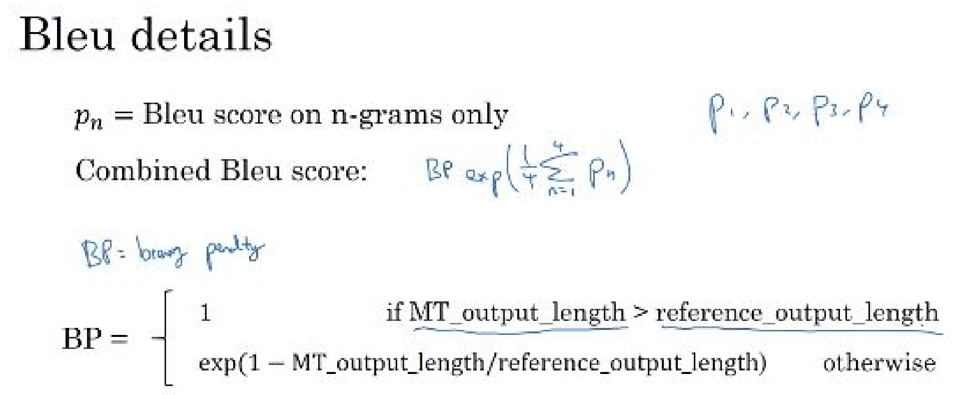
n-gram

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

Papineni et. al., 2002. Bleu: A method for automatic evaluation of machine translation]

Andrew Ng

- Lastly, let's put it all together. By convention, we compute $P_1,\ P_2,\ P_3\ and\ P_4$ and we combine them using the formula below. It is basically the average



Bleu details

p_n = Bleu score on n-grams only

P_1, P_2, P_3, P_4

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

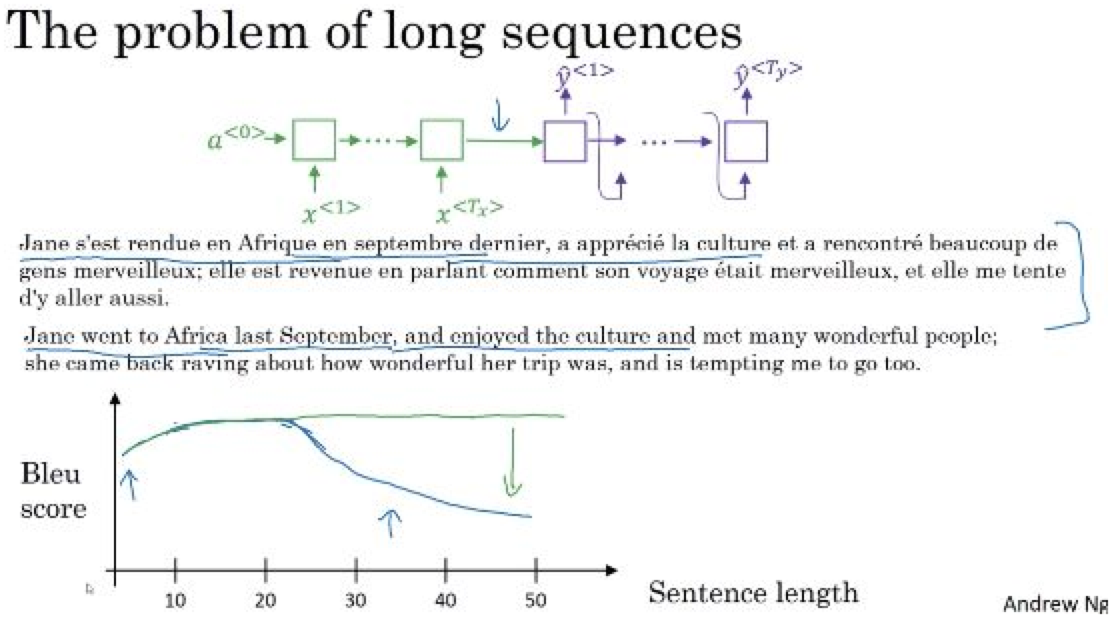
BP = brevity penalty

$$BP = \begin{cases} 1 & \text{if MT\_output\_length > reference\_output\_length} \\ \exp(1 - \text{MT\_output\_length/reference\_output\_length}) & \text{otherwise} \end{cases}$$

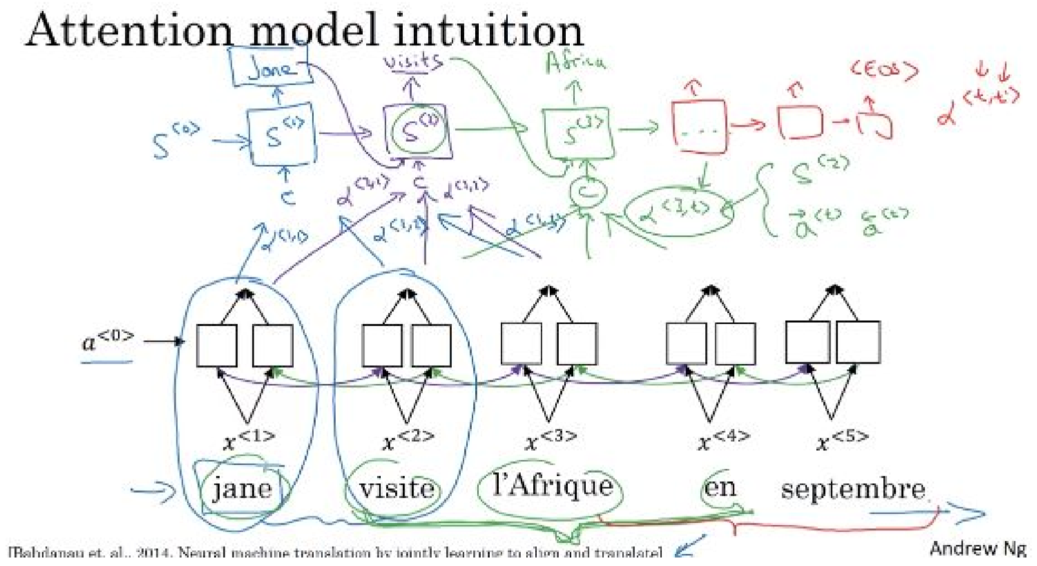
The formula is basically the average augmented by the BP, the brevity penalty. BP, as its name implies, is a penalty applied to machine translations that are shorter than the references. It has two cases, which are shown above. The formulas are:
$BP = 1\ if\ MTOutputLength > referenceOutputLength$ and
$BP = exp(1 - referenceOutputLength/MTOutputLength)$.

Attention Model Intuition

- For most of this week we have been using the encoder/decoder model. This model works well for short sentences but struggles with long sentences. To remedy that we are going to look at the attention model. Given the long sentence below a human will translate it a few chunks at a time. The Bleu score for the encoder/decoder model decreases as the sentence length increase. Attention model does better. Let's look at it.

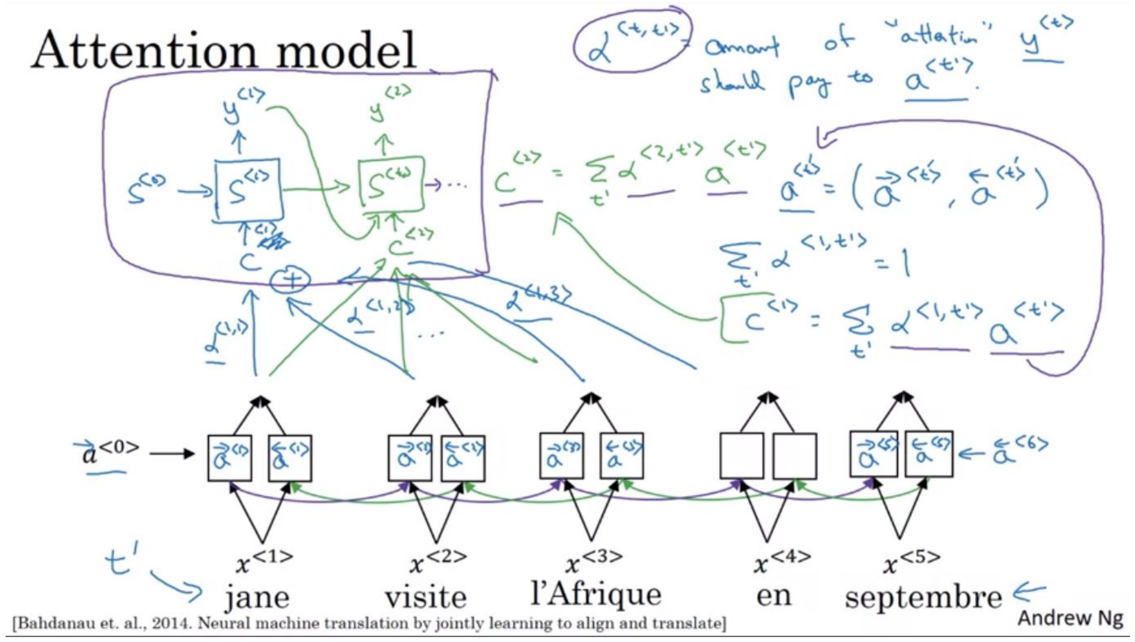

The problem of long sequences
$a^{<0>}$
$x^{<1>}$
$x^{<T_x>}$
$\hat{y}^{<1>}$
$\hat{y}^{<T_y>}$
Jane s'est rendue en Afrique en septembre dernier, a apprécié la culture et a rencontré beaucoup de gens merveilleux; elle est revenue en parlant comment son voyage était merveilleux, et elle me tente d'y 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.
Bleu score
10
20
30
40
50
Sentence length
Andrew Ng

- What we do is we start with our input sentence below. We then use a bidirectional RNN to compute a rich set of features for each word in the input sentence. We then use another RNN to generate the english translation. Instead of using a we are going to use S to denote the activations and hidden states of this second RNN. So we generate $S^{<1>}$ and the question we face is what part of the input sentence should we be looking at so we can generate the output word? Here's where the attention model enters. It computes parameters alpha (α), also known as attention weights, that will tell us how much attention should we pay to each piece of information from the first RNN. For example, for $S^{<1>}$ we look at $\alpha^{<1,1>}$, $\alpha^{<1,2>}$, $\alpha^{<1,3>}$. We do the same for each layer until we finish. One more detail, next session we will see that say, for step $S^{<3>}$, the amount of attention we should pay to the french word at time t is defined by the activations $a^{<t>}$, the previous activations $a^{\leftarrow<t>}$ and the previous state, $S^{<2>}$ in this case.


Attention model intuition
jane
visite
l'Afrique
en
septembre
[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]
Andrew Ng

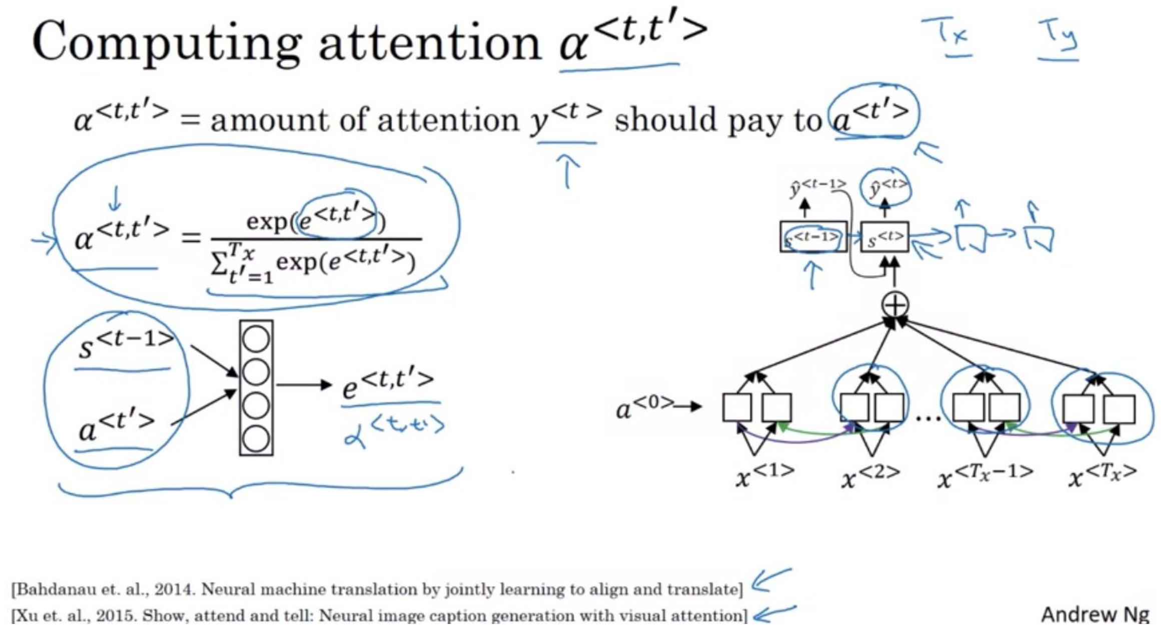
Attention Model

- Let's now look at how we implement an attention model. We start with our input sentence in french and our bidirectional RNN will generate a list of features that we will use in the next step. We then start our unidirectional RNN where we take a context $C^{<t>}$ which is defined for say, the first layer as $C^{<1>} = \sum_{t'} \alpha^{<1,t'>} * a^{<t'>}$. What we are saying is the context is the sum of the product of $\alpha^{<1,t'>}$ (the amount of attention $y^{<t>}$ should pay to $* \ a^{<t'>}$) and the features learned in the previous step.


Attention model
amount of "attention" $y^{<t>}$ should pay to $a^{<t'>}$
$C^{<2>} = \sum_{t'} \alpha^{<2,t'>} a^{<t'>}$
$a^{<t'>} = (\overrightarrow{a}^{<t'>}, \overleftarrow{a}^{<t'>})$
$\sum_{t'} \alpha^{<1,t'>} = 1$
$C^{<1>} = \sum_{t'} \alpha^{<1,t'>} a^{<t'>}$
jane
visite
l'Afrique
en
septembre
[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]
Andrew Ng

We define $\alpha^{<t,t'>}$ as the amount of attention the $y^{<t>}$ should pay to $*$ $a^{<t'>}$. The attention weights are all non negative and its sum will be one, in other words $\sum_{t'} \alpha^{<1,t'>} = 1$

- How do we compute $\alpha^{<t,t'>}$? We know it stands for the amount of attention $y^{<t>}$ should pay to $a^{<t'>}$. The formula is shown below. Notice there is a factor e. How do we compute them? We use a small neural network, usually a 1 layer neural network that has as inputs $s^{<t-1>}$, the hidden state of the previous layer and $a^{<t'>}$, the features from that state, and the output is the factor $e^{<t,t'>}$. The neural network will learn the function and gradient descent will optimize it.



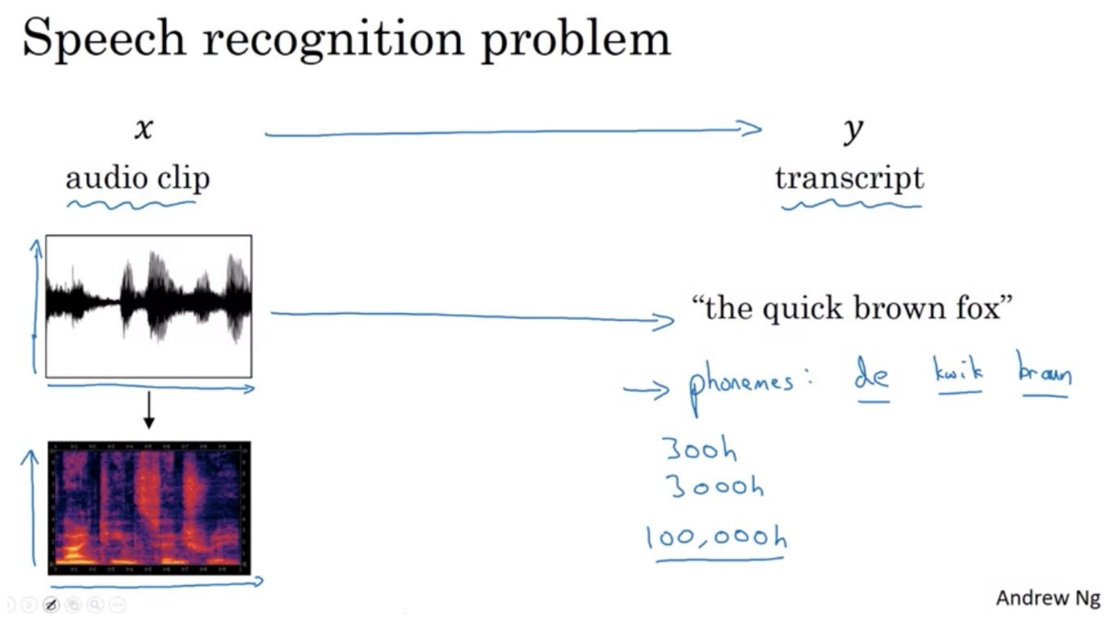
Computing attention $\alpha^{<t,t'>}$
T_x
T_y
$\alpha^{<t,t'>}$ = amount of attention $y^{<t>}$ should pay to $a^{<t'>}$
$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x}\exp(e^{<t,t'>})}$
$s^{<t-1>}$
$a^{<t'>}$
$e^{<t,t'>}$
$\alpha^{<t,t'>}$
$\hat{y}^{<t-1>}$
$\hat{y}^{<t>}$
$s^{<t-1>}$
$s^{<t>}$
$a^{<0>}$
$x^{<1>}$
$x^{<2>}$
$x^{<T_x-1>}$
$x^{<T_x>}$
[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]
[Xu et. al., 2015. Show, attend and tell: Neural image caption generation with visual attention]
Andrew Ng

One downside of this algorithm is that it has a quadratic cost. If we have T_x words as input and T_y as output then the cost is $T_x * T_y$

- Our programming exercise will use the attention model to solve the date normalization problem.

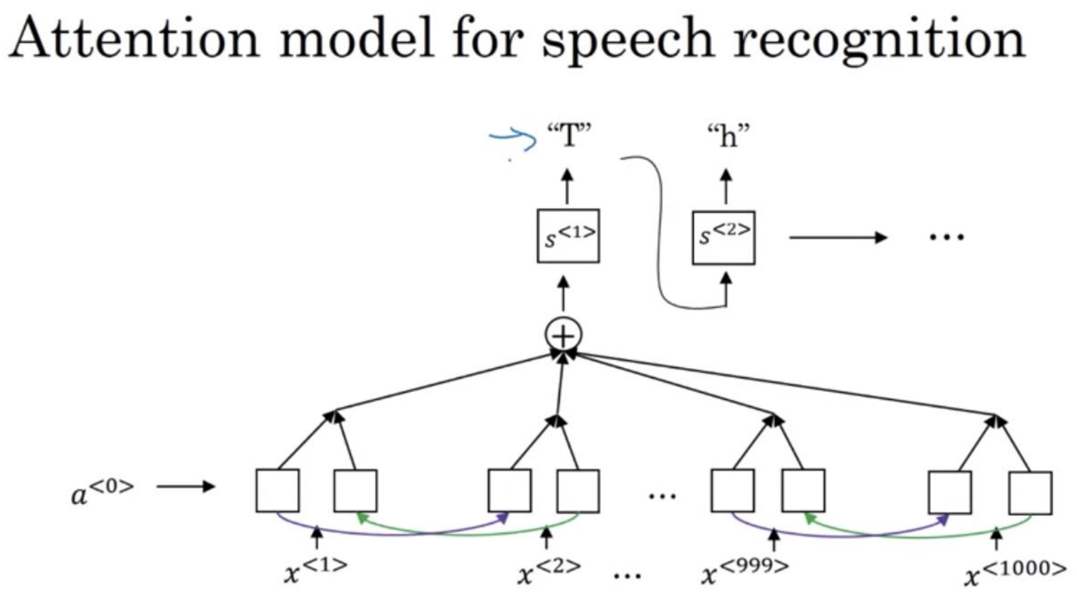
Speech Recognition

- In this session we are going to look at how speech recognition systems are built. For these problems we have an audio clip x and our job is to produce a transcript y. Once upon a time, before the advent of deep learning, we would build such a system with hand-engineered basic units called phonemes. This is no longer necessary; now we can take an audio clip and directly produce the transcript. What made this possible, in part, was the ability to train our algorithms with larger data sets. In this day and age, speech recognition applications are trained with over 10,000 and up to 100,000 hours of audio.



Speech recognition problem

- We can apply an attention model where the input is different time frames of the audio clip and the output is the transcript.

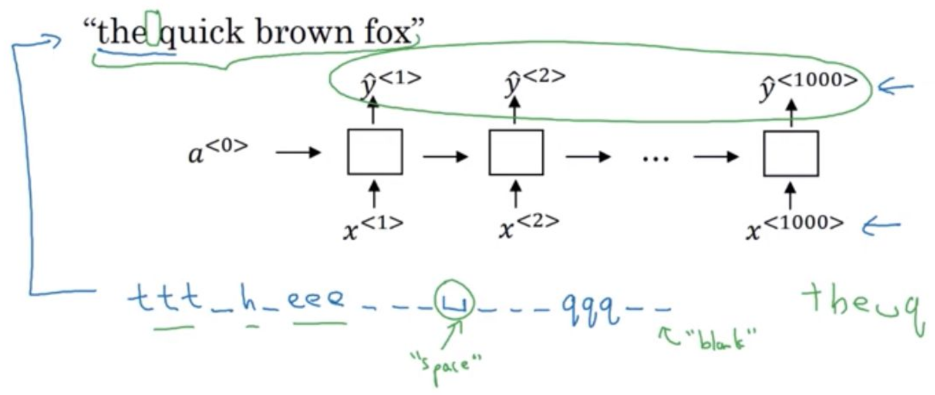


Attention model for speech recognition

- Another tool we can use is CTC cost where CTC stands for connectionist temporal classification. Assume we have an audio clip that has the phrase 'the quick brown fox'. We have a neural network (unidirectional in this example but in practice you use something more complex like an LSTM model) that has the same number of inputs and outputs. In practice this is not often the case because an audio clip will have bigger inputs than outputs. However, we can get around this limitation by forcing the neural network to output a long sequence using CTC.

CTC cost for speech recognition

(Connectionist temporal classification)



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

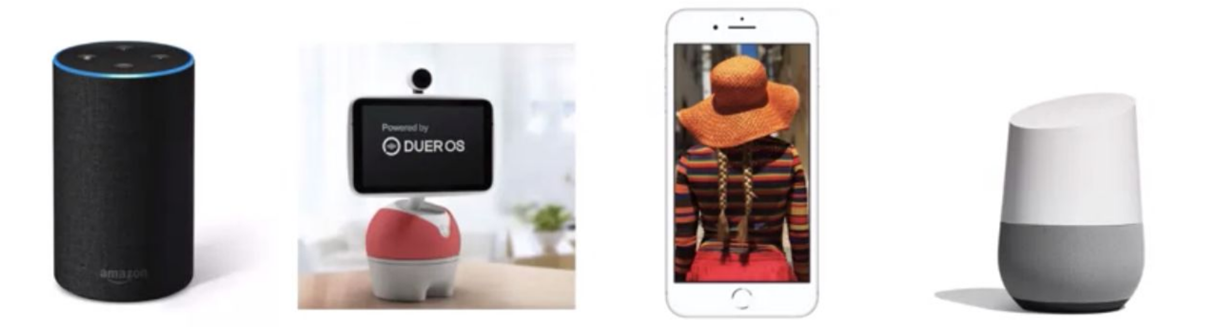
[Graves et al., 2006. Connectionist Temporal Classification: Labeling unsegmented sequence data with recurrent neural networks] Andre

What CTC will do is repeat characters across the neural network and will then collapse repeated characters that are not separated by the blank character.

Trigger Word Detection

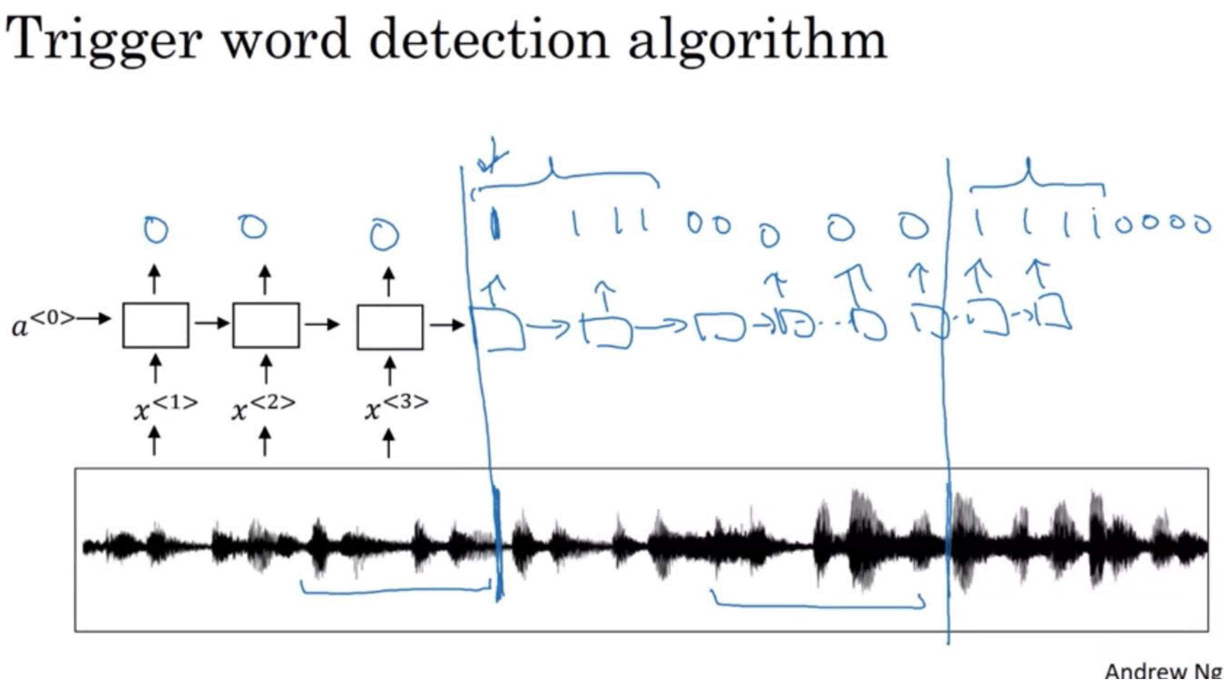
- In this last session we look at how to build a trigger word detection. Trigger words are used to 'wake up' systems like Amazon Echo or Google Home.

What is trigger word detection?



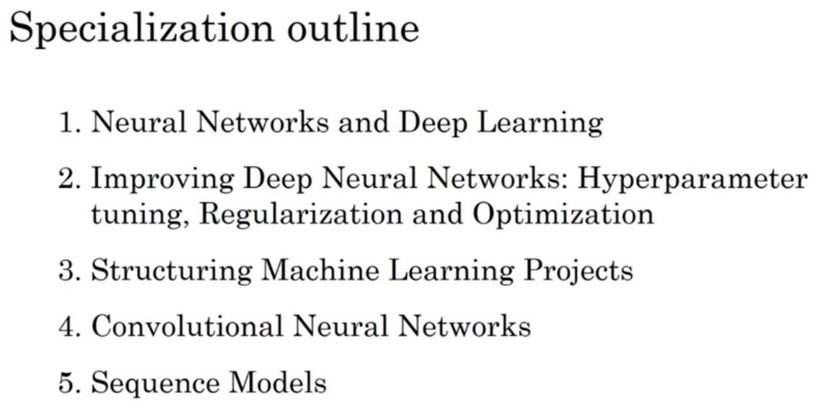
Amazon Echo (Alexa) | Baidu DuerOS (xiaodunihao) | Apple Siri (Hey Siri) | Google Home (Okay Google)

- There is no standard for a trigger word detection algorithm so what is shown below is one example. What we do is we take an audio clip, generate features x1, x2 and so on and those are the inputs to the RNN. After that we just need to define the target labels y. When we detect the trigger word, in the training set we set the labels =0 for everything before and = 1 right after that. Below are 2 examples of such a case. One problem with this algorithm is that it creates an unbalanced training set (way more zeros than ones). One hacky thing we can do to make the model easy to train is to generate more 1's for a fixed period of time and then revert back to zeros.



Conclusion

- We have been in quite a journey together.We learned all these things below.



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

Deep Learning is a superpower, it allows us to make a car drive itself, we can make a computer see, we can synthesize music and so on. Now that we have it we can use it to make this world a better place! :-)

Learnings from 1st programming exercise: Neural Machine Translation with Attention

- Machine translation models can be used to map from one sequence to another. They are useful not just for translating human languages (like French->English) but also for tasks like date format translation.
- An attention mechanism allows a network to focus on the most relevant parts of the input when producing a specific part of the output.
- A network using an attention mechanism can translate from inputs of length T_x to outputs of length y where T_x and T_y can be different.
- You can visualize attention weights $\alpha^{<t,t'>}$ to see what the network is paying attention while generating each output

Learnings from 2nd programming exercise: Trigger Word detection

- 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