Sequence Models

dzlabs

September 2018

1 Week 1 - Recurrent Neural Networks

1.1 Recurrent Neural Networks

1.1.1 Why sequence models

Wide sequence of applications for Sequence models.

1.1.2 Notation

Motivation example: Entity Name recognition, given a sentence X, return y a binary victor with 1 on entities word and 0 otherwise.

- $x^{(i)<t>}$ to notate the TIF element in the ith training example.
- $T_x^{(i)}$ the length of the ith training example
- $y^{(i)<t>}$ tif element in the ith output example
- $T_y^{(i)}$ the length of the ith output example

To represent a word in a sentence. First define a vocabulary / Dictionary which is a list of the words that you will use in your representations. Dictionary size in commercial applications can reach 30k, for the largest companies 1M.

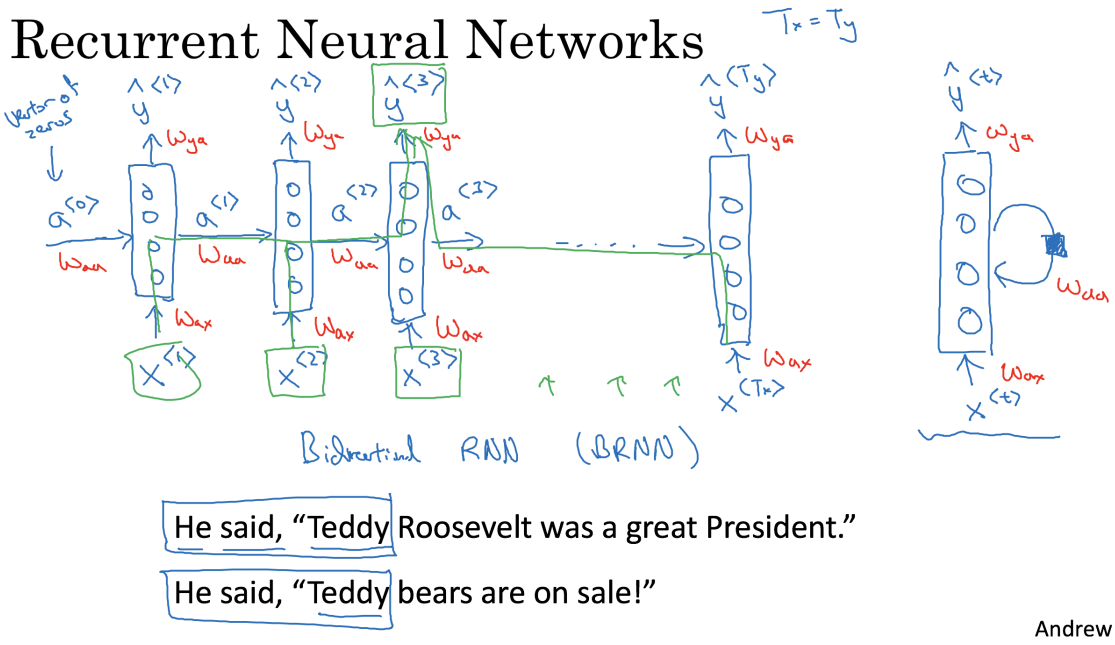
One hot representation to repsent the words, e.g. the representation of $x < 1 >$ (i.e. Harry in the example) is a victor (also known one hot) with 1 at the index of Harry in the dictionnay, 0 everywhere else. You can use work $< UNK >$ for words which are not in your dictionnary.

1.1.3 Recurrent Neural Network Model

For mapping X to y, why not a standard network? the problems is that:

- Inputs, outputs can be different lengths in different examples.
- A naive architecture like this does not share features learned across different positions of text.

Like with CNN we would like things learned for one part of the image to generalize quickly to other parts of the image, and we like a similar effects for sequence data as well.



If you're reading a sentence from right to left, you pass the first word to the hidden layer, then for the second word you will also passes on the activation of the previous layer. In the first layer, an activation vector of 0s is used. This architecture uses only the previous words, but not the later words, which is a problem. This can be addressed by Bidirectional RNN (BRNN) which we will see later.

The activation function for the hidden units can be tahn or ReLU. For activation of the output, it can sigmoid if binary, or softmax in case of a k-way classification.

- $a^{<0>} = 0$
- $a^{<1>} = g(w_{aa}a^{<0>} + w_{ax}x^{<1>} + b_a)$
- $\hat{y}^{<1>} = g(w_{ya}a^{<1>} + b_y)$

More generaly:

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

To simplify the previous general equations, stack the vectors $a^{<t-1>}$, $x^{<t>}$ together, and compress the parameters into one vector parameter:

- $a^{<t>} = g(W_a[a^{<t-1>}, x^{<t>}])$
- $w_a = [w_{aa}, w_{ax}]$
- $\hat{y}^{<t>} = g(w_y a^{<t>} + b_y)$

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

1.1.4 Backpropagation through time

It's called through time as you calculate from the right to the left i a decresing time index (back in time), while forward goes from left to right in the increase time.

Standard logistic regression loss, also called the cross entropy loss.

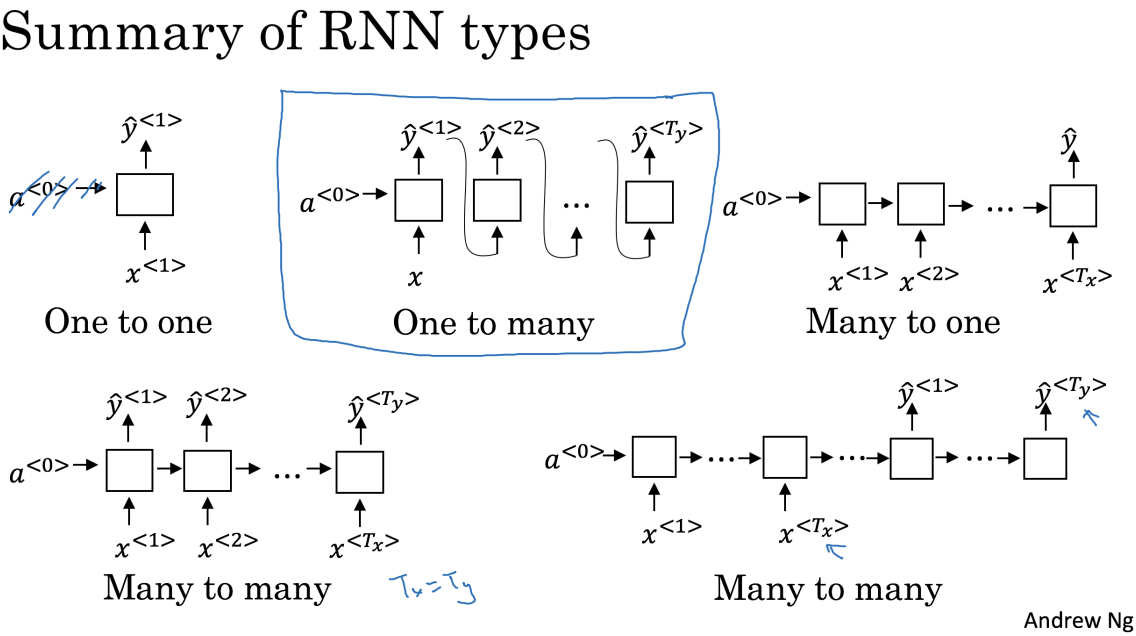
$$\mathcal{L}^{<t>}(\hat{y}^t, y^t) = -y^t \log \hat{y}^t - (1 - y^t) \log(1 - \hat{y}^t)$$

The loss for the entire sequence:

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

1.1.5 Different types of RNNs

- many-to-many: input sequence has many inputs as a sequence and the outputs sequence is also has many outputs. e.g. music generation, starting from note for instance.
- many-to-one: it inputs many words and then it just outputs one number, e.g. sentiment classification
- one-to-one: a starndard NN, convered in the first course of this sequence

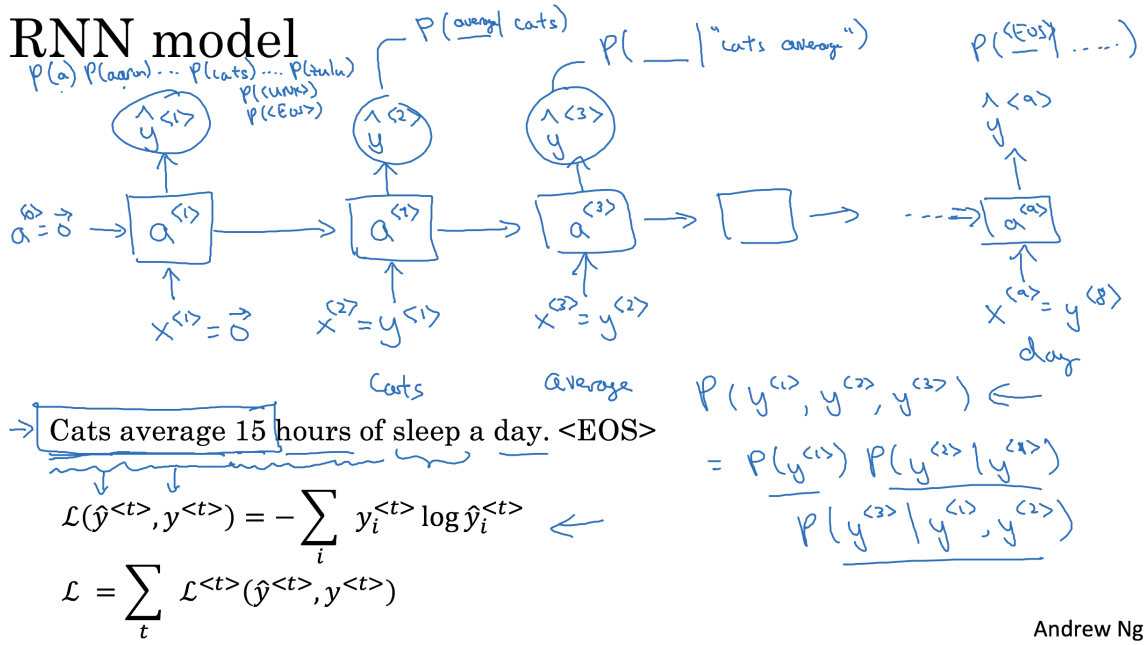


1.1.6 Language model and sequence generation

Language model, choose between two sentences based on a probability to decide which one is more likely. A language modeal, given any sentence, does tell you what is the probability of that particular sentence.

Build a language model with an RNN, training set: large corpus of english text. For the model to capture the end of sentence, add to each training sentence the $< EOS >$ token at its end.

RNN model: What that's going to be $y^{<1>}$: this step has a soft max it's trying to predict. What is the probability of any word in the dictionary? it would be a 10,000 way soft max output, or 10,002, if we call unknown word and end of the sentence is two additional tokens.



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

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

The output of the previous layer in passed to the next one, ie $x^{<t>} = y^{<t>}$, this RNN learns to generate a word a time by looking at all the previous ones.

Given a three words sentence, $y^{<1>}$, $y^{<2>}$, $y^{<3>}$. Predict the probability to see this words in sequence:

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

It turns out one of the most fun things you could do with a language model is to sample sequences from the model.

1.1.7 Sampling novel sequences

After training a **sequence model**, one of the ways to informally get a sense of what is learned is to have a sample novel sequences.

After training a sequence model, input an X=0, then from the output $y^{<1>}$ which is 10k way softmax, then sample one word of this distribution and pass it to the next step/timestamp, etc, until you get to the end. To detect the end of sentence, wait until you see $< EOS >$ in the output. This is how to generate a randomly sampled sentence.

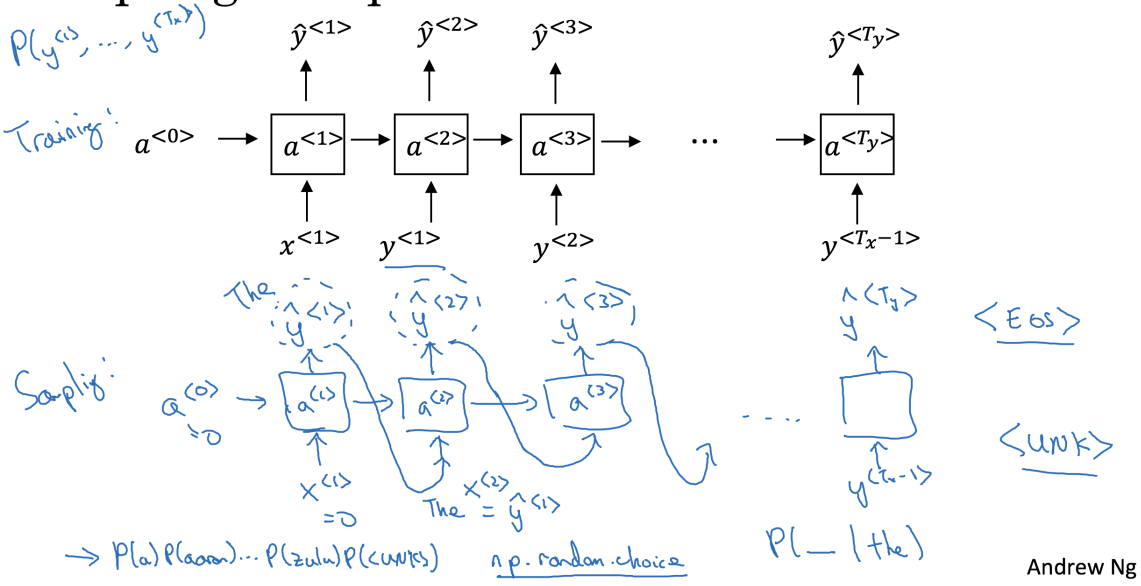


Figure 1: Sampling a sequence from a trained RNN

Character-level language mode: In this case, the vocabulary is a sequence of alphabet characters, e.g. Vocabulary = [a, b, c, ..., z, space, dot, comma, 0, ..., 9, A, ..., Z].

The main disadvantage of the character level language model is that:

- you end up with much longer sequences. So many english sentences will have 10 to 20 words but may have many, many dozens of characters.
- And so character language models are not as good as word level language models at capturing long range dependencies between how the the earlier parts of the sentence also affect the later part of the sentence.
- And character level models are also just more computationally expensive to train.

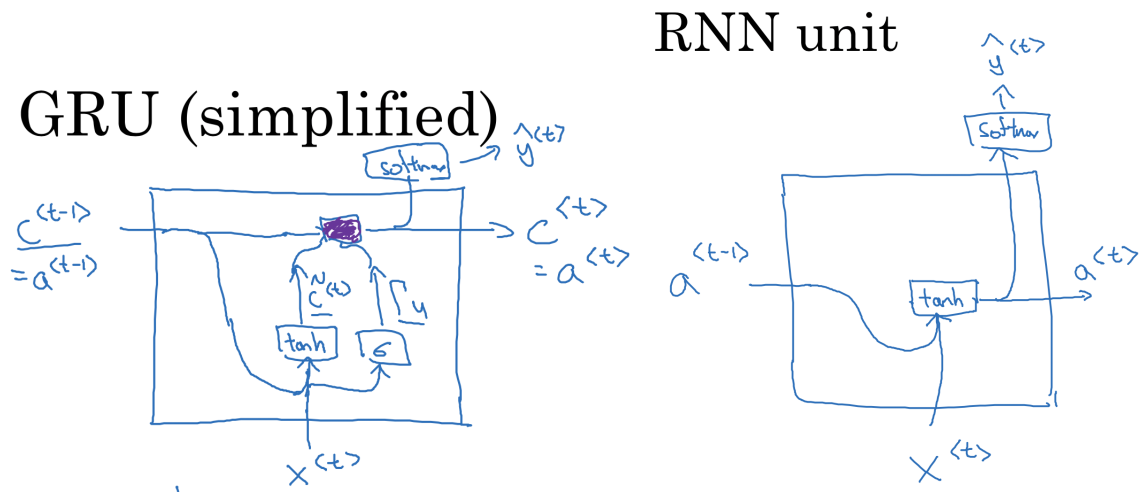
1.1.8 Vanishing gradients with RNNs

One of the problemns of basic RNN is that it runs into vanishing gradients and exploding problems: An example of long term dependencies in a laguage: when outputing 'cat' later we should have 'was', similarly when outputting 'cats' we should have 'were'. In a deep NN, gradient have hard time propagating back, i.e. errors associated with later timestamps have hard time impacting erros in earlier timestamps. RNN output is influced by locality, i.e, timestamps nearby. For this, RRN are not good at capturing **Long range dependencies**.

Similarly, if gradient becomes too big **Expliding gradients** (i.e. NaN) as function of number of layers, then apply gradient clipping. i.e. look at your gradient vectors, and if it is bigger than some threshold, re-scale some of the gradient vector so that is not too big. So there are clips according to some maximum value.

1.1.9 Gated Recurrent Unit (GRU)

It's a modification to the RNN hidden layer that makes it much better capturing long range connections and helps a lot with the vanishing gradient problems.



(a) Gradient Recurrent Unit

(b) Recurrent basic unit

Figure 2: RNN units

Memory cell at t, $C^{<t>} = a^{<t>}$ then $\hat{C}^{<t>} = \tanh(w_c[C^{<t-1>}, x^{<t>}] + b_c)$ The important idea is to have a **gate**, alway zero or one, in practice it's calculated using a sigmoid function, which has a role to decide when to use a word. With 'u' stands for update, the formula is:

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

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

`Full GRU` Many different possible versions of how to design these units, to try to have longer range connections, to try to have more the longer range effects and also address vanishing gradient problems. And the GRU is one of the most commonly used versions. e.g, full GRU formula is:

$$\hat{C}^{<t>} = \tanh(w_c[\Gamma_r * C^{<t-1>}, x^{<t>}] + b_c) \tag{2}$$

Figure 3: The candidate for replacing the memory cell

$$\Gamma_u = \sigma(w_u[C^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_r = \sigma(w_r[C^{<t-1>}, x^{<t>}] + b_r)$$

Use the gate Γ_u to decide whether or not to update the memory cell.

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

1.1.10 Long Short Term Memory (LSTM)

A more powerful and general GRU version. It has three gates instead of two an place them in different places:

- Γ_u the update gate
- Γ_f the forget gate
- Γ_o the output gate

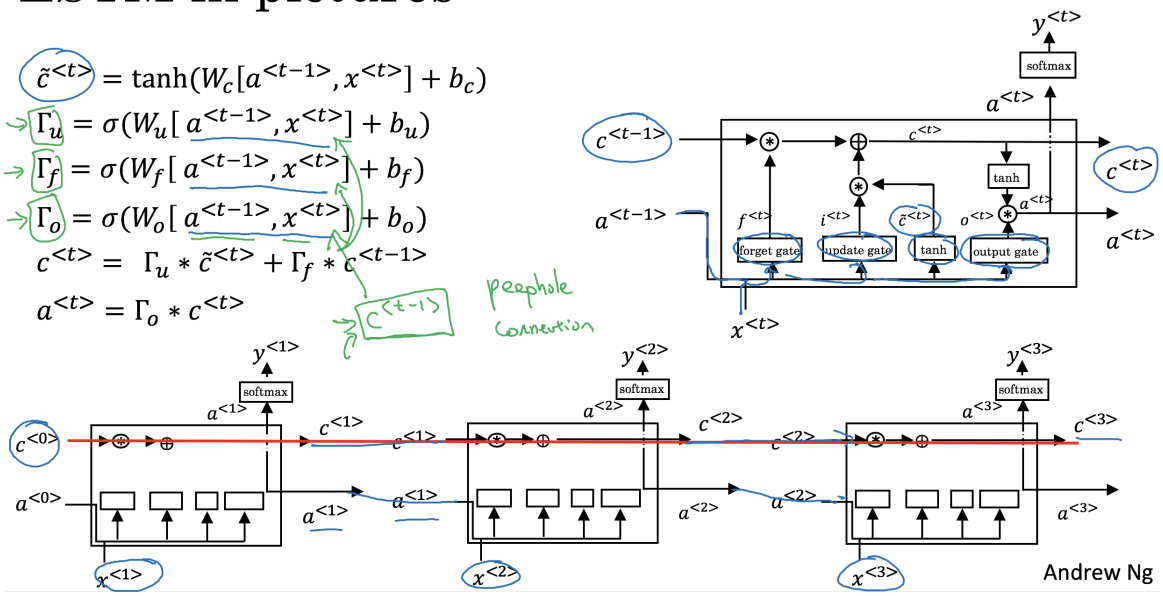


Figure 4: LSTM

Peephole connection mean that c_t minus one is used to affect the gate value as well.

- The advantage of the GRU is that it's a simpler model and so it is actually easier to build a much bigger network, it only has two gates, so computationally, it runs a bit faster. So, it scales the building somewhat bigger models.
- the LSTM is more powerful and more effective since it has three gates instead of two. If you want to pick one to use, I think LSTM has been the historically more proven choice.

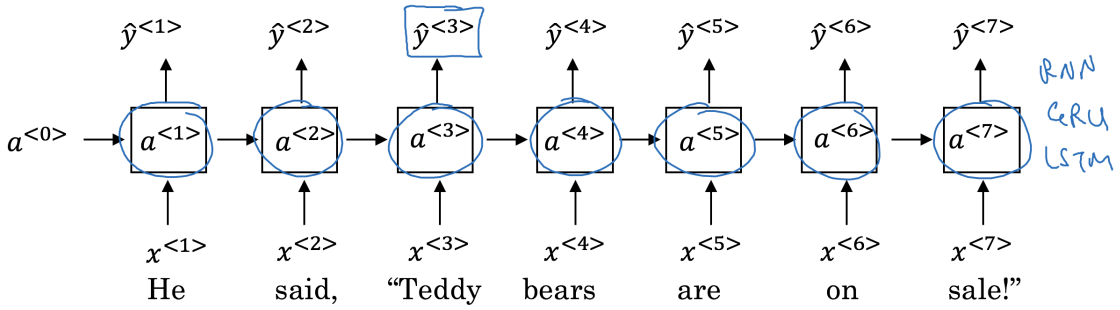


Figure 5: Example of network of RNN,GRU,LSTM blocks

1.1.11 Bidirectional RNN

In a BRNNinformation from $x^{[1]}$ can flow forward to $\hat{y}^{<3>}$ through $\vec{a}^{<1>}$, $\vec{a}^{<2>}$ and $\vec{a}^{<3>}$. Information can also flow backward from $x^{[4]}$ to $\overleftarrow{a}^{[4]}$ to backward $\overleftarrow{a}^{[y]}$ and finally $\hat{y}^{<3>}$.

This allows the prediction of output to take information from the past as well as from the future.

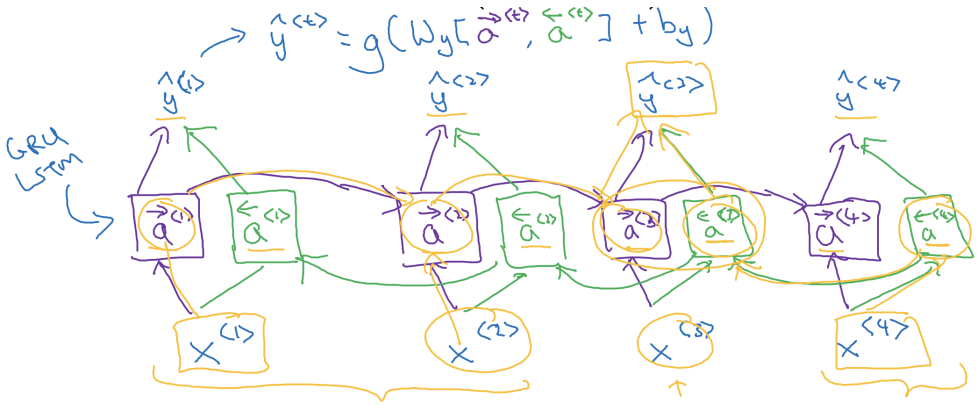


Figure 6: Example acyclic graph for BRNN

In this figure, the blocks can be basic RNN blocks, GRU or LSTM.

The disadvantage is that you need the entire sequence before you actually start processing, e.g. in speech recognition, you need the person to stop talking to start processing.

1.1.12 Deep RNNs

To learn complex functions, it is useful to stack many layers of RNN.

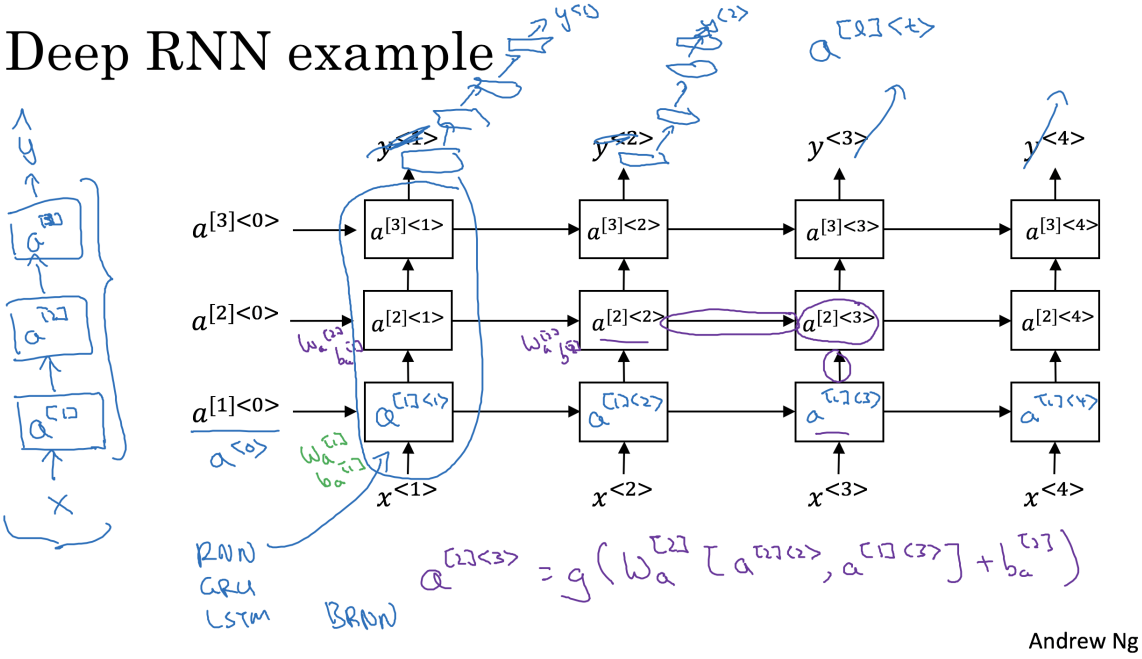


Figure 7: Example of DeepRNN with 3 layers

Example of how to calculate activation function:

$$a^{[2]<3>} = g(w_a^{[2]}[a^{[2]<2>}, a^{[1]<3>}] + b_a^{<3>})$$

1.2 Practice questions

1.2.1 QUIZ - Recurrent Neural Networks

1. Suppose your training examples are sentences (sequences of words). Which of the following refers to the j^{th} word in the i^{th} training example?

- $x^{(i)<j>}$ (XConsider this RNN:)
- $x^{<i>(j)}$
- $x^{(j)<i>}$

- $x^{<j>(i)}$

2. Consider this RNN: This specific type of architecture is appropriate when:

- $T_x = T_y$ (X)
- $T_x < T_y$
- $T_x > T_y$
- $T_x = 1$

3. To which of these tasks would you apply a many-to-one RNN architecture? (Check all that apply).

- Speech recognition (input an audio clip and output a transcript)
- Sentiment classification (input a piece of text and output a 0/1 to denote positive or negative sentiment) (X)
- Image classification (input an image and output a label)
- Gender recognition from speech (input an audio clip and output a label indicating the speaker's gender) (X)

4. You are training this RNN language model. At the t^{th} time step, what is the RNN doing? Choose the best answer.

- Estimating $P(y^{<1>}, y^{<2>}, ..., y^{<t-1>})$
- Estimating $P(y^{<t>})$
- Estimating $P(y^{<t>} \mid y^{<1>}, y^{<2>}, ..., y^{<t-1>})$ (X)
- Estimating $P(y^{<t>} \mid y^{<1>}, y^{<2>}, ..., y^{<t>})$

5. You have finished training a language model RNN and are using it to sample random sentences, as follows: What are you doing at each time step t?

- (i) Use the probabilities output by the RNN to pick the highest probability word for that time-step as $\hat{y}^{<t>}$. (ii) Then pass the ground-truth word from the training set to the next time-step.
- (i) Use the probabilities output by the RNN to randomly sample a chosen word for that time-step as $\hat{y}^{<t>}$. (ii) Then pass the ground-truth word from the training set to the next time-step.
- (i) Use the probabilities output by the RNN to pick the highest probability word for that time-step as $\hat{y}^{<t>}$. (ii) Then pass this selected word to the next time-step.
- (i) Use the probabilities output by the RNN to randomly sample a chosen word for that time-step as $\hat{y}^{<t>}$. (ii) Then pass this selected word to the next time-step. (X)

6. You are training an RNN, and find that your weights and activations are all taking on the value of NaN ("Not a Number"). Which of these is the most likely cause of this problem?

- Vanishing gradient problem.
- Exploding gradient problem. (X)
- ReLU activation function g(.) used to compute g(z), where z is too large.
- Sigmoid activation function g(.) used to compute g(z), where z is too large.

7. Suppose you are training a LSTM. You have a 10000 word vocabulary, and are using an LSTM with 100-dimensional activations $a^{<t>}$. What is the dimension of Γ_u at each time step?

- 1
- 100

- 300
- 10000 (X)

8. Here're the update equations for the GRU.

Alice proposes to simplify the GRU by always removing the Γ_u. I.e., setting $\Gamma_u = 1$. Betty proposes to simplify the GRU by removing the Γ_r. I. e., setting $\Gamma_r = 1$ always. Which of these models is more likely to work without vanishing gradient problems even when trained on very long input sequences?

- Alice's model (removing Γ_u), because if $\Gamma_r \approx 0$ for a timestep, the gradient can propagate back through that timestep without much decay.
- Alice's model (removing Γ_u), because if $\Gamma_r \approx 1$ for a timestep, the gradient can propagate back through that timestep without much decay. (X)
- Betty's model (removing Γ_r), because if $\Gamma_u \approx 0$ for a timestep, the gradient can propagate back through that timestep without much decay.
- Betty's model (removing Γ_r), because if $\Gamma_u \approx 1$ for a timestep, the gradient can propagate back through that timestep without much decay.

9. Here are the equations for the GRU and the LSTM:

From these, we can see that the Update Gate and Forget Gate in the LSTM play a role similar to _____ and _____ in the GRU. What should go in the the blanks?

- Γ_u and $1 - \Gamma_u$ (X)
- Γ_u and Γ_r
- $1 - \Gamma_u$ and Γ_u
- Γ_r and Γ_u

10. You have a pet dog whose mood is heavily dependent on the current and past few days' weather. You've collected data for the past 365 days on the weather, which you represent as a sequence as $x^{<1>}, ..., x^{<365>}$. You've also collected data on your dog's mood, which you represent as $y^{<1>}, ..., y^{<365>}$. You'd like to build a model to map from $x \rightarrow y$. Should you use a Unidirectional RNN or Bidirectional RNN for this problem?

- Bidirectional RNN, because this allows the prediction of mood on day t to take into account more information.
- Bidirectional RNN, because this allows backpropagation to compute more accurate gradients.
- Unidirectional RNN, because the value of $y^{<t>}$ depends only on $x^{<1>}, ..., x^{<t>}$, but not on $x^{<t+1>}, ..., x^{<365>}$ (X)
- Unidirectional RNN, because the value of $y^{<t>}$ depends only on $x^{<t>}$, and not other days' weather.

2 Week 2 - Natural Language Processing & Word Embeddings

2.1 Introduction to Word Embeddings

2.1.1 S

o far we were repsenting words with vocabulary, and 1-hot representation (binary vector). This treats each word as a thing in itself and does not allow the algorithm to learn relationships between words. This is because the in-product between any word is 0, it does not know that orange and apple are similar words.

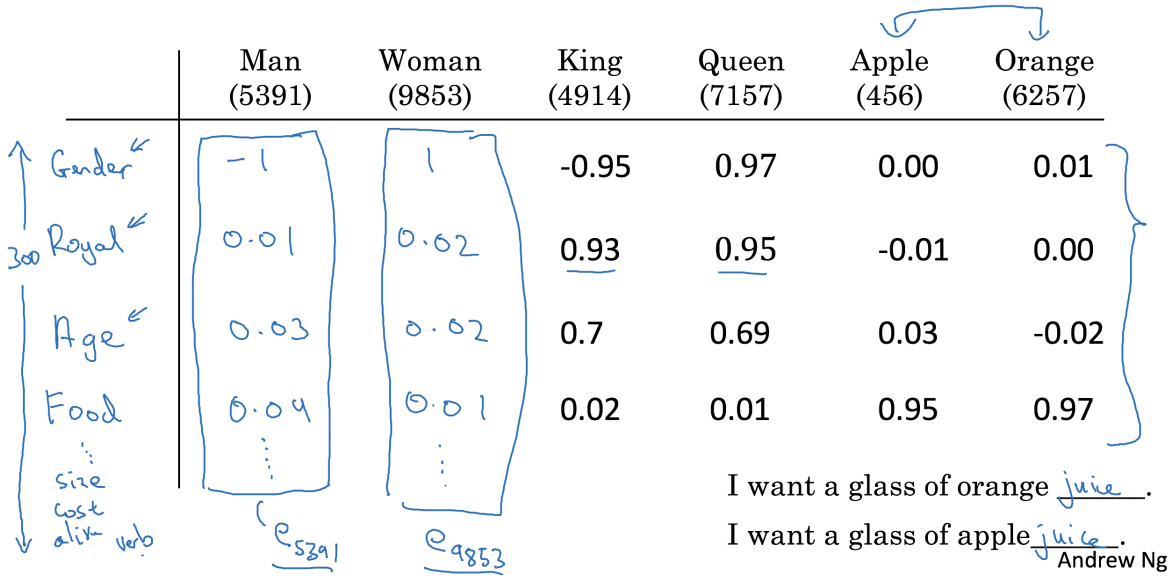


Figure 8: Featurized representation: word embedding:

Instead, as in the picture, represent a word using high dimensional feature vector of values for each feature (e.g. gender, age, food). This allows to generalize better and learn a vectorized representation of words.

One of the algorithm, is `t-SNE`. A word is embeded in a 300 dimensional space, this is why it's called word embedding.

2.1.2 Using word embeddings

Transfer learning and word embeddings

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

Word embeddings (also called encoding)

- It has been useful for named entity recognition, for text summarization, for co-reference resolution, for parsing. These are all maybe pretty standard NLP tasks.
- It has been less useful for language modeling, machine translation, especially if you're accessing a language modeling or machine translation task for which you have a lot of data just dedicated to that task.

2.1.3 Properties of word embeddings

Embeddings also help with analogies. Substracting the vectors for two words (say man and women, or kind and queen), capture the differences, in this case gender. The resulting vector should have approximately 0s everywhere except for the feature gender.

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

Figure 9: Analogies using word vectors

Find word w:

$$\arg\, max_w \ \text{Sim}(e_w, e_{king} - e_{man} + e_{woman}) \tag{3}$$

The mostly used similarity function is the **Cosine similarity**, cosine of the angle between two vectors:

$$Sim(u, v) = \frac{u^T v}{||u||_2 \, ||v||_2} \tag{4}$$

It;s also possible to use the **Euclidean** distance, which is used most for dis-similarity.

2.1.4 Embedding matrix

Let's start to formalize the problem of learning a good word embedding. When you implement an algorithm to learn a word embedding, what you end up learning is an **embedding matrix**.

The goal is to lear the Embedding matrix `E`

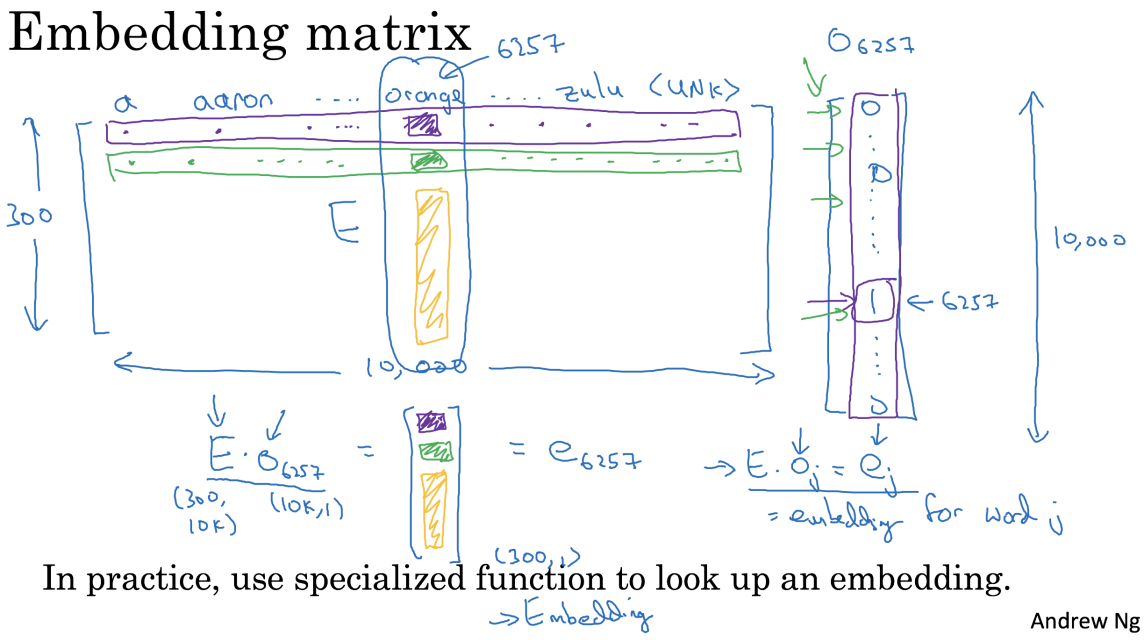


Figure 10: Embedding matrix

The multiplication of $E * O_{6257}$ is computation expensive as this O vector is one-hot vector all zeros but at the row for word 'Orange'. In practice we sue a specialized function to lookup an embedding, i.e. the columns that corresponds to the word Orange.

2.2 Learning Word Embeddings: Word2vec & GloVe

Algorithms for learning the Embedding matrix `E`.

2.2.1 Learning word embeddings

The parameters that need to be learned by the algorith, are: E, $W^{<1>}$, $b^{<1>}$.

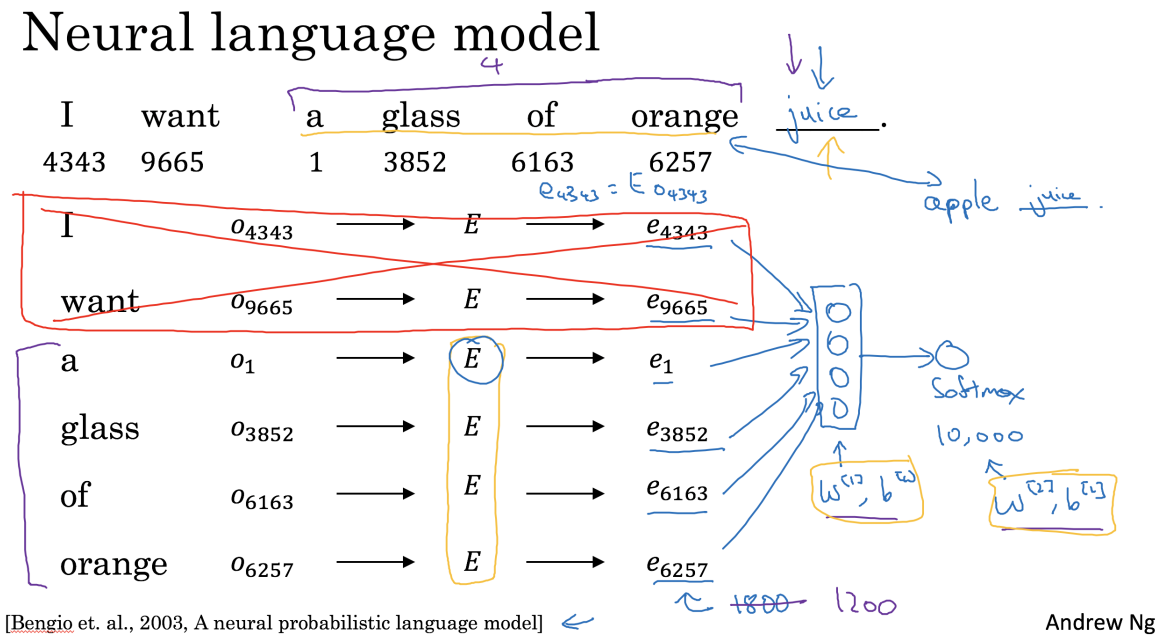


Figure 11: Learning word embedding

Context (the last three allows to learn embedding):

- last 4 words in the sentence,
- 4 words on left and right,
- last 1 word,
- nearby word.

  In the next video, you'll see how using even simpler context and even simpler learning algorithms to mark from context to target word, can also allow you to learn a good word embedding.

2.2.2 Word2Vec

A simple algorithm with efficient computation.

Model Vocab size = 10,000. Content c x="orange" (word index: 6257) ==> Traget t y="juice" (word index: 4834)

$$o_c -> E -> e_c = E * o_c -> o = softmax(e_c) -> \hat{y}$$

Softmax: θ_t parameter associated with ouput t.

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

Loss function:

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

Primary problem of this model is the computation speed, as you have to carry the sum of 10k in the softmax. One way to improve this is by using **Hierarchical Softmax** that gives you $log|v|$.

Different heuristics can be used to build the tree, to have common words in the top of the tree so that you can access them in few steps, while less common words are burried deep in the tree.

To sample the context c, you can sample randomly from your corpus, but you can heuritics so that $P(c)$ for sampling is not uniform, you don't want to have same probability for frequent words against uncommon words.

2.2.3 Negative Sampling

Defining a new supervised learning problem:

To generate the training set:

1. pick a context word (say orange) and a target word (say juice), then gives this row (e.g. orange juice) label 1.
2. For k times, take same context word, then pick random word from dictionnary (e.g. of, the, book, etc.) and label all of them with 0. i.e. negative examples.
3. Then, the supervised learning algorithm, will take X (context and target word) and try to predict the output label y.

To choose K, it can be from 5 to 20, for lagest dataset chose smaller values (2 to 5).

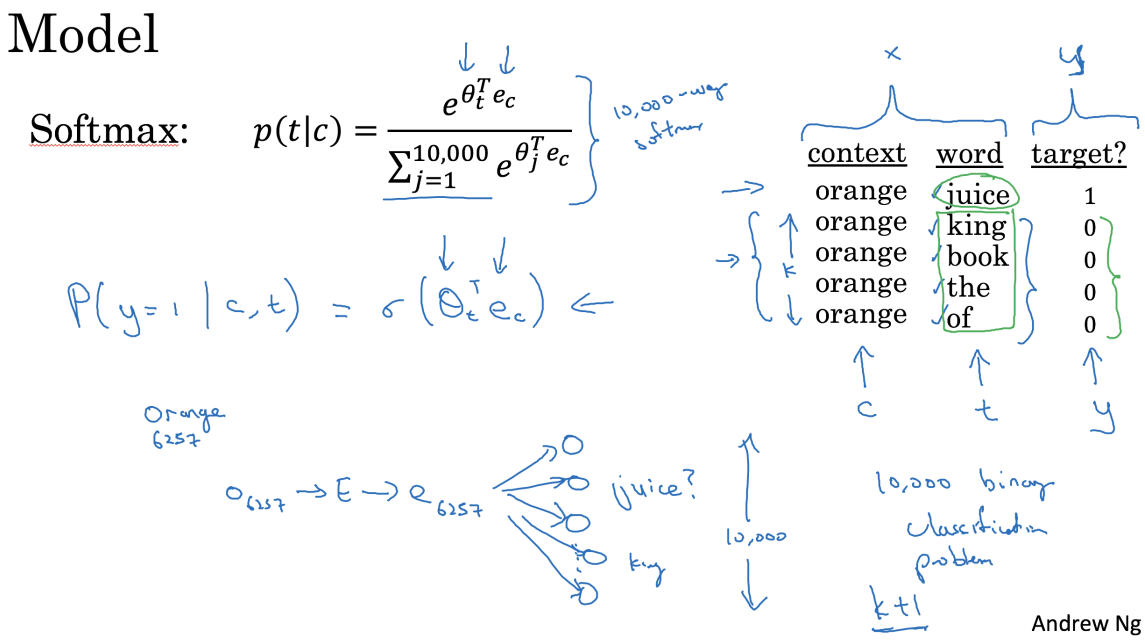


Figure 12: Model of the supervised logistic regression

So think of this as having 10,000 binary logistic regression classifiers, but instead of training all 10,000 way Softmax on every iteration, we're only going to train five of them. We're going to train the one responding to the actual target word we got and then train 4 randomly chosen negative examples. This in case $k == 4$.

One word to sample the words (i.e. $P(w_i)$) for the negative samples is with words distribution but you will endup with common words like a, the, of. An alternative is to use the frequency f:

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

2.2.4 GloVe word vectors

GloVe (global vectors for word representation), having Context c and taget t:

$$X_{ij} = \#\text{ times i appears in context of j}$$

Model: GloVe tries to optimize the following:

$$\text{mimimize} \sum_{i=1}^{10000} \sum_{j=1}^{10000} f(X_{ij})(\theta_i^T e_j + b_i + b'_j - \log X_{ij})^2$$

- $f(X_{ij}$ is a weithing term is 0 if $X_{ij} = 0$, it is used to avoid calculating log of 0.
- θ_i, θ_j are symmetric

- $e_w^{(final)} = \frac{e_w + \theta_w}{2}$

The features detected by this algorithm are possible be non interpretable by humans, i.e. not just gender, age. But rather the learned features can be a combinations of say gender and age, etc.

2.3 Applications using Word Embeddings

2.3.1 Sentiment Classification

If you train this algorithm, you end up with a pretty decent sentiment classification algorithm and because your word embeddings can be trained from a much larger data set, this will do a better job generalizing to maybe even new words now that you'll see in your training set, such as if someone else says, "Completely absent of good taste, good service, and good ambiance" or something, then even if the word "absent" is not in your label training set, if it was in your 1 billion or 100 billion word corpus used to train the word embeddings, it might still get this right and generalize much better even to words that were in the training set used to train the word embeddings but not necessarily in the label training set that you had for specifically the sentiment classification problem. So that's it for sentiment classification, and I hope this gives you a sense of how once you've learned or downloaded from online a word embedding, this allows you to quite quickly build pretty effective NLP systems.

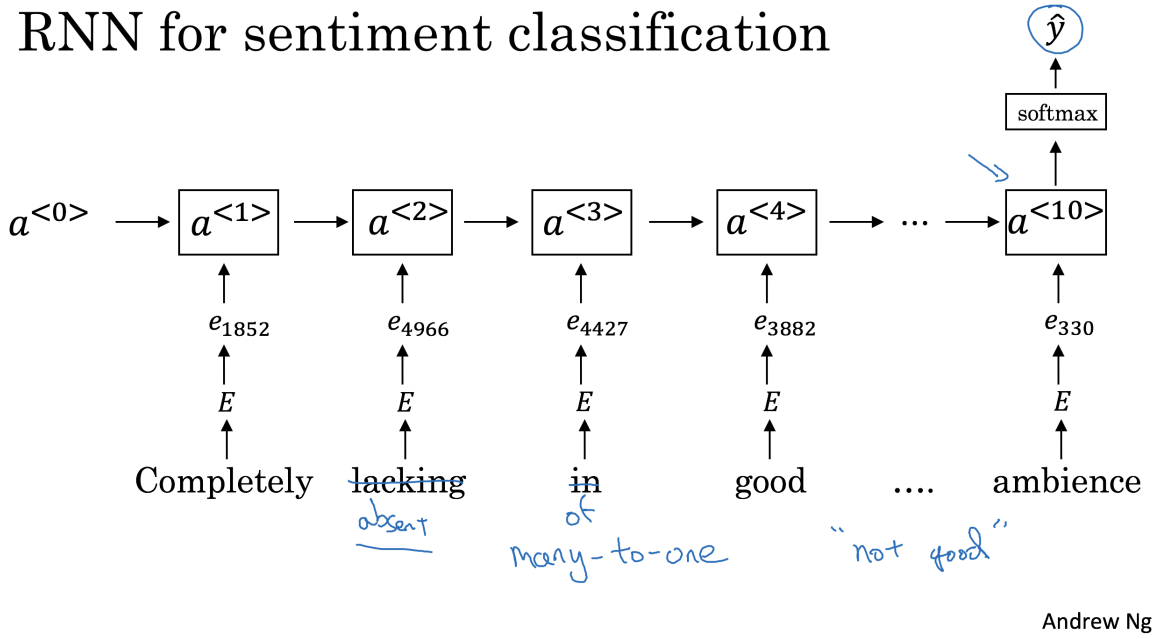


Figure 13: RNN for sentiment analysis

2.3.2 Debiasing word embeddings

Some of the ideas of dimishing bias. In this case, it's about gender, ethnicity bias. Word embeddings can reflect the gender, ethnicity, age, sexual orientation, and other biases of the text used to train the model. You want wordds like Doctor to be gender neutral.

First step, Identifying bias direction:

- $e_{he} - e_{she}$
- $e_{male} - e_{femal}$
- ...

Then average them (or simialar idea to PCA), to detect bias direction. Second step, neutralize: for every word that is not definitional (not words like grandmother which is female, grandfather which is male), project to get rid of bias Third step, equalize pairs, e.g. grandmother/girl should be equalized to grandfather/boy.

To decide fo what words are definitial, a linear classifier can tell you what words can be classified as definitional vs. not.

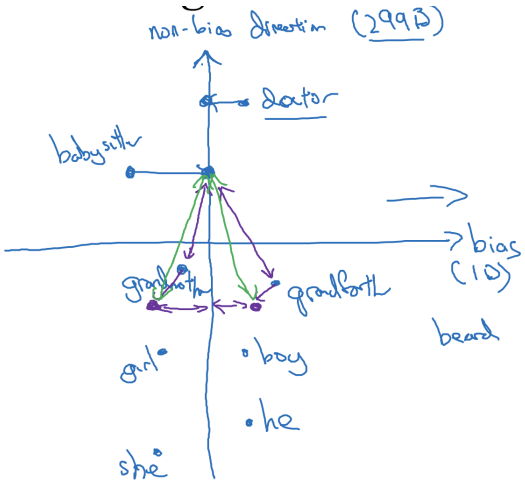


Figure 14: Addressing bias in word embeddings

2.4 Practice questions

2.4.1 QUIZ - Natural Language Processing & Word Embeddings

1. Suppose you learn a word embedding for a vocabulary of 10000 words. Then the embedding vectors should be 10000 dimensional, so as to capture the full range of variation and meaning in those words.

- True
- False (X) The dimension of word vectors is usually smaller than the size of the vocabulary. Most common sizes for word vectors ranges between 50 and 400.

2. What is t-SNE?

- A linear transformation that allows us to solve analogies on word vectors
- A non-linear dimensionality reduction technique (X 2)
- A supervised learning algorithm for learning word embeddings (X 1) No
- An open-source sequence modeling library

3. Suppose you download a pre-trained word embedding which has been trained on a huge corpus of text. You then use this word embedding to train an RNN for a language task of recognizing if someone is happy from a short snippet of text, using a small training set.

Then even if the word "ecstatic" does not appear in your small training set, your RNN might reasonably be expected to recognize "I'm ecstatic" as deserving a label y = 1y=1.

- True (X) Yes, word vectors empower your model with an incredible ability to generalize. The vector for "ecstatic would contain a positive/happy connotation which will probably make your model classified the sentence as a "1".
- False

4. Which of these equations do you think should hold for a good word embedding? (Check all that apply)

- $e_{boy} - e_{girl} \approx e_{brother} - e_{sister}$ (X)
- $e_{boy} - e_{girl} \approx e_{sister} - e_{brother}$
- $e_{boy} - e_{brother} \approx e_{girl} - e_{sister}$ (X)
- $e_{boy} - e_{brother} \approx e_{sister} - e_{girl}$

5. Let E be an embedding matrix, and let o_{1234} be a one-hot vector corresponding to word 1234. Then to get the embedding of word 1234, why don't we call $E * o_{1234}$ in Python?

- It is computationally wasteful. (X) Yes, the element-wise multiplication will be extremely inefficient.
- The correct formula is $E^T * o_{1234}$
- This doesn't handle unknown words (¡UNK¿).
- None of the above: calling the Python snippet as described above is fine.

6. When learning word embeddings, we create an artificial task of estimating $P(target \mid context)$. It is okay if we do poorly on this artificial prediction task; the more important by-product of this task is that we learn a useful set of word embeddings.

- True (X)
- False

7. In the word2vec algorithm, you estimate $P(t \mid c)$, where t is the target word and c is a context word. How are t and c chosen from the training set? Pick the best answer.

- c and t are chosen to be nearby words.
- c is a sequence of several words immediately before t. (X 2)
- c is the sequence of all the words in the sentence before t.
- c is the one word that comes immediately before t. (X 1)

8. Suppose you have a 10000 word vocabulary, and are learning 500-dimensional word embeddings. The word2vec model uses the following softmax function:

$$P(t|c) = \frac{e_t^{\theta^T} e_c}{\sum_{t'=1}^{10000} e_{t'}^{\theta^T} e_c}$$

Which of these statements are correct? Check all that apply.

- θ and e_c are both 500 dimensional vectors. (X)
- θ_t and e_c are both 10000 dimensional vectors.
- θ_t and e_c are both trained with an optimization algorithm such as Adam or gradient descent. (X)
- After training, we should expect θ_t to be very close to e_c when t and cc are the same word.

9. Suppose you have a 10000 word vocabulary, and are learning 500-dimensional word embeddings.The GloVe model minimizes this objective:

$$min \sum_{i=1}^{10000} \sum_{j=1}^{10000} = f(X_{ij})(\theta_i^T e_j + b_i + b'_j - \log X_{ij})^2$$

Which of these statements are correct? Check all that apply.

- θ_i and e_j should be initialized to 0 at the beginning of training.
- θ_i and e_j should be initialized randomly at the beginning of training. (X)
- X_{ij} is the number of times word i appears in the context of word j. (X)
- The weighting function $f(.)$ must satisfy $f(0) = 0$. (X) The weighting function helps prevent learning only from extremely common word pairs. It is not necessary that it satisfies this function.

10. You have trained word embeddings using a text dataset of m_1 words. You are considering using these word embeddings for a language task, for which you have a separate labeled dataset of m_2 words. Keeping in mind that using word embeddings is a form of transfer learning, under which of these circumstance would you expect the word embeddings to be helpful?

- $m_1 >> m_2$ (X)
- $m_1 << m_2$

3 Week 3 - Sequence models & Attention mechanism

3.1 Various sequence to sequence architectures

3.1.1 Basic Models

Sequence models are useful from translation to speech recognition.

Sequence to sequence model: Given a pair of sentences in frech and english this model works well. A similar architecture works also well for image capturing. This is how it works: have a CNN that given an image detects a set of features, this can be the encoder network that will inputs one feature at a time to the RNN.

3.1.2 Picking the most likely sentence

As depcted in the figure, they have similar architectures except that instead of all zero $a^{<0>}$, the encoder network is trying to figure out the set of features that will be passed to the language model.

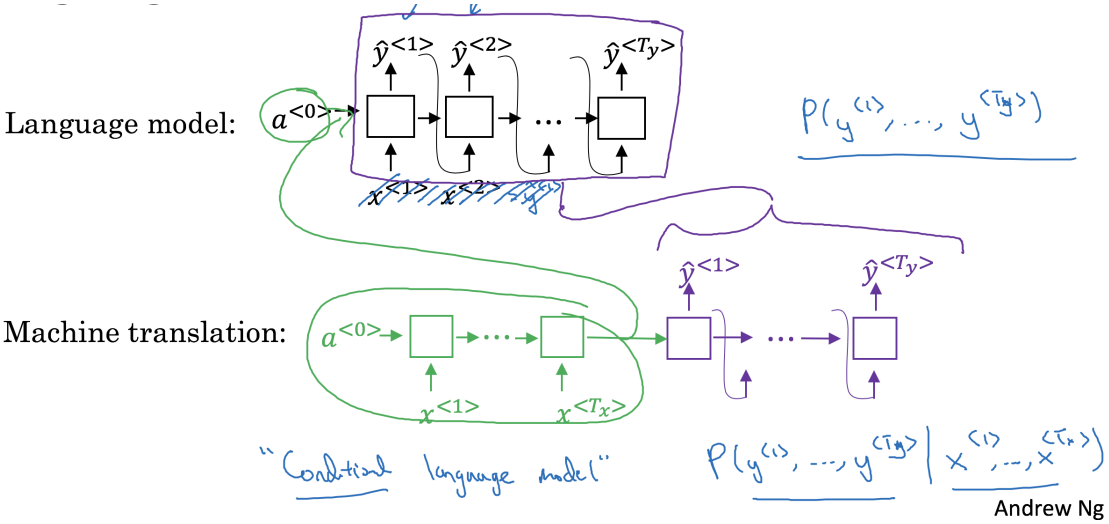


Figure 15: Machine translation can be thought as building a conditional language model

We don;t want to sample randomly, what you're looking for is a sentence that maximize

$$\arg\max_{y^1,...,y^{T_y}} P(y^1,...,y^{T_y}|x)$$

Why not a **Greedy Search** to solve this problem? i.e. pick a first word $P(\hat{y}^{<i>}|x)$, then pick second mosltly likely word, etc.

What do you want is to pick the entire sequence at once while maximizing the probability $P(\hat{y}^{<1>},...,\hat{y}^{<T_y>}|x)$ as it's not always optimal to pick a word at once. What you can use is an approximate search algorithm, as scanning the entire space.

3.1.3 Beam Search

Beam Search has a parameter B `beam width` (e.g. 3), it first selects the B most likely words and feed them into a segment of the encoding net, not really sure.

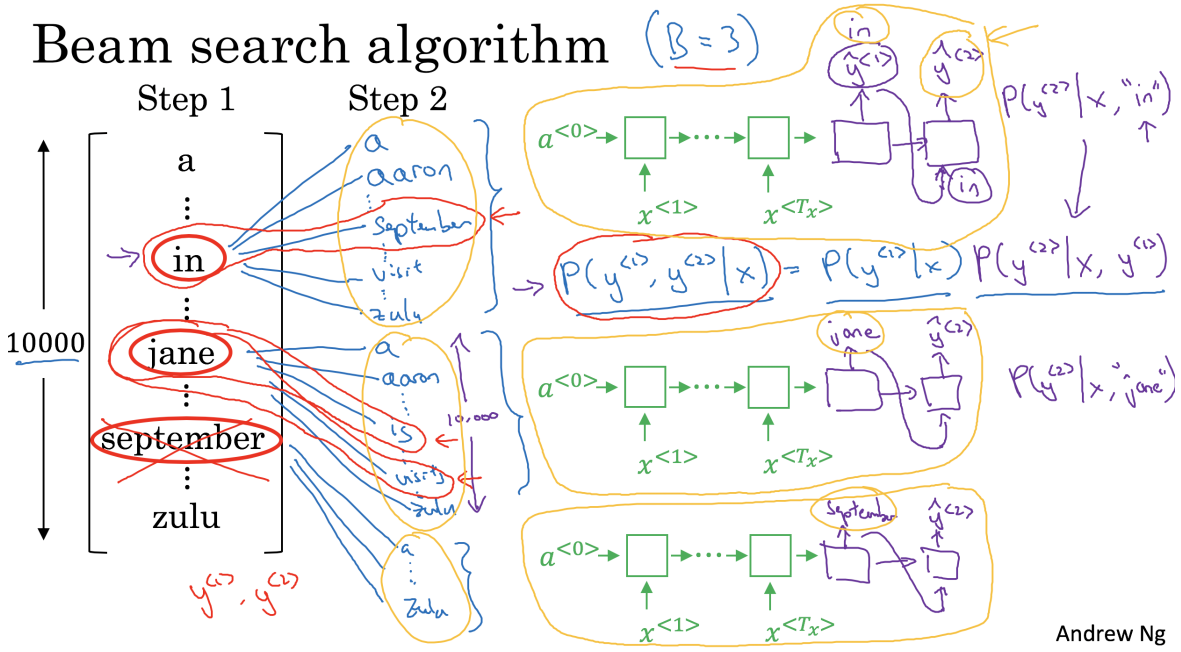


Figure 16: Beam Search Algorithm

It will try to predict the probability of the third word to select, i.e. $\hat{y}^{<3>}$ given x (the french sentence to translate) and the english translation so far (e.g. 'in september'), i.e. $P(y^{<3>}|x, 'in september')$.

3.1.4 Refinements to Beam Search

Little changes to make Beam works better: Length normalization

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

All these number are too small, multiplying them return a really small number (that may underflow). In practice, we rather use the log function thus

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

Another change to the above equation to futher optmize it, is to normalize by the legth of the sentence to avoid penalizing long sentences (where α is usually 0.7):

$$\arg\max_y \frac{1}{T_y^\alpha} \Pi_{t=1}^{T_y} \log P(y^{<y>}|x, y^{<1>}, ..., y^{<t-1>})$$

- If B is large, you tend to consider a lot of possibibilites and thus it's slower
- If B is small you endup with worst results but faster runtime

In production system, B is usually 10, at 100 it's too large for prod system. In research you can see up to 1000 as they aim to get the best performance.

> 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 $max_y P(y|x)$

3.1.5 Error analysis in beam search

Error analysis on beam search

- Human: Jane visits Africa in September. (y^*)
- Algorithm: Jane visited Africa last September. ($\hat{y}$)
- 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: $P(y^*|x) <= P(\hat{y}|x)$: y^* is a better translation than $\hat{y}$. But RNN predicted $P(y^*|x) <= P(\hat{y}|x)$. Conclusion: *RNN model is at fault.*

Figures out what faction of errors are "due to"" Beam Search vs. RNN model

3.1.6 Bleu Score (optional)

For a given french sentence, we may have multiple correct translations. One way to measure how good is this machine translation is by the Blue Score. Also called, Bilingual Evaluation understudy. A substitute for human evaluation

Clipping the count, is reducing the count of a unigram or bigram to the max count of it's appearance in one of Reference sentences.

$$P_1 = \frac{\sum_{unigram \in \hat{y}} Cout_{clip}(unigram)}{sum_{unigram \in \hat{y}} Cout(unigram)} \tag{5}$$

n-grams

$$P_n = \frac{\sum_{n-gram \in \hat{y}} Cout_{clip}(n-gram)}{sum_{n-gram \in \hat{y}} Cout(n-gram)} \tag{6}$$

So the BP, or the brevity penalty, is an adjustment factor that penalizes translation systems that output translations that are too short.

3.1.7 Attention Model Intuition

Till now we were using an architecture of encoding-decoding NN.

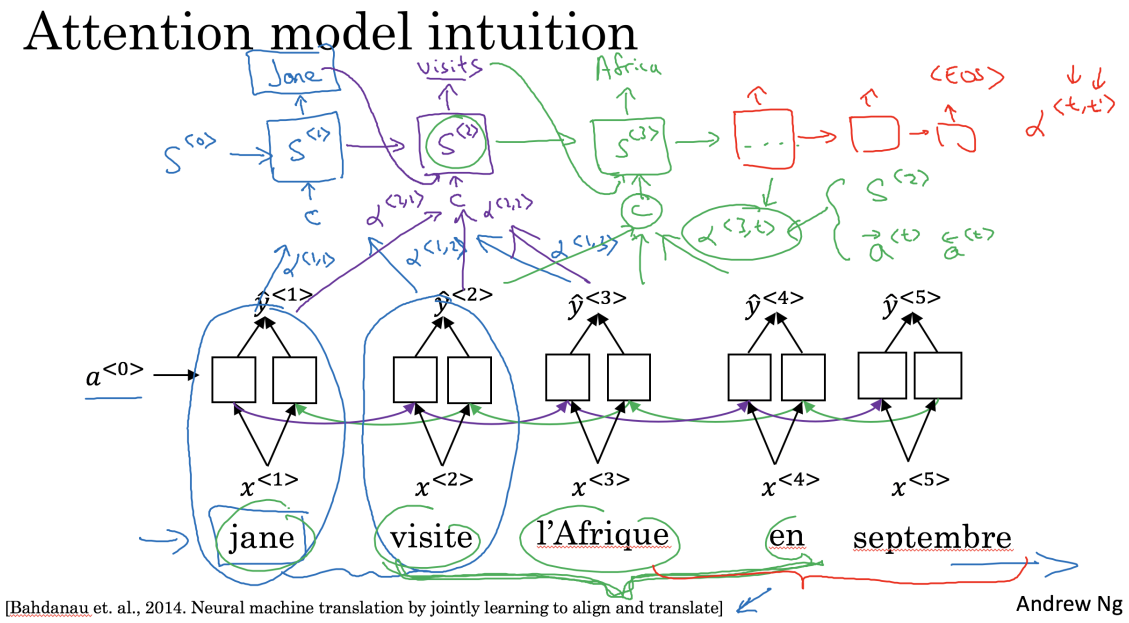


Figure 17: Attention model intuition

3.1.8 Attention Model

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

Computing attention $\alpha^{<t,t'>}$:

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

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

This algorithms take quadric time to compute it's parameters.

3.2 Speech recognition - Audio data

3.2.1 Speech recognition

Attention algorithms.

CTC cost for speech recognition:

> if you have 10 seconds of audio and your features come at a 100 hertz so 100 samples per second, then a 10 second audio clip would end up with a thousand inputs. Right, so it's 100 hertz times 10 seconds, and so with a thousand inputs. But your output might not have a thousand alphabets, might not have a thousand characters.

To address this issue, CTC inserts black words between characters.

3.2.2 Trigger Word Detection

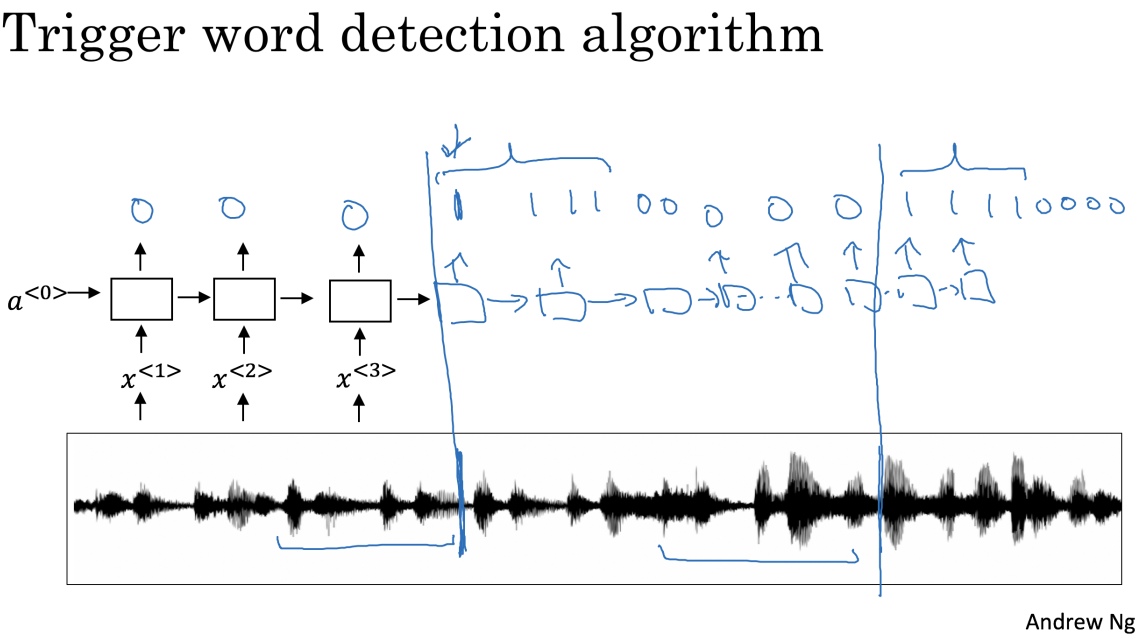


Figure 18: Trigger word detection algorithm

> to wrap up in this course on sequence models you learned about rnns including both gr use and LS TMS and then in the second week you learned a lot about word embeddings and how they learn representations of words and then in this week you learned about the attention model as well as how to use it to process audio data

3.3 Conclusion

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

3.4 Practice questions

3.4.1 QUIZ - Sequence models & Attention mechanism

1. Consider using this encoder-decoder model for machine translation.

This model is a "conditional language model" in the sense that the encoder portion (shown in green) is modeling the probability of the input sentence x.

- True (X 1)
- False (X 2)

2. In beam search, if you increase the beam width BB, which of the following would you expect to be true? Check all that apply.

- Beam search will run more slowly. (X)
- Beam search will use up more memory.
- Beam search will generally find better solutions (i.e. do a better job maximizing $P(y \mid x)$ (X 1)
- Beam search will converge after fewer steps.

3. In machine translation, if we carry out beam search without using sentence normalization, the algorithm will tend to output overly short translations.

- True (X)
- False

4. Suppose you are building a speech recognition system, which uses an RNN model to map from audio clip xx to a text transcript y. Your algorithm uses beam search to try to find the value of y that maximizes $P(y \mid x)$.

On a dev set example, given an input audio clip, your algorithm outputs the transcript $\hat{y}$ = "I'm building an A Eye system in Silly con Valley.", whereas a human gives a much superior transcript y^* = "I'm building an AI system in Silicon Valley."

According to your model,

$$P(\hat{y} \mid x) = 1.09 * 10^-7$$

$$P(y^* \mid x) = 7.21 * 10^-8$$

Would you expect increasing the beam width B to help correct this example?

- No, because $P(y^* \mid x) \leq P(\hat{y} \mid x)$ indicates the error should be attributed to the RNN rather than to the search algorithm. (X)
- No, because $P(y^* \mid x) \leq P(\hat{y} \mid x)$ indicates the error should be attributed to the search algorithm rather than to the RNN.
- Yes, because $P(y^* \mid x) \leq P(\hat{y} \mid x)$ indicates the error should be attributed to the RNN rather than to the search algorithm.
- Yes, because $P(y^* \mid x) \leq P(\hat{y} \mid x)$ indicates the error should be attributed to the search algorithm rather than to the RNN.

5. Continuing the example from Q4, suppose you work on your algorithm for a few more weeks, and now find that for the vast majority of examples on which your algorithm makes a mistake, $P(y^* \mid x) > P(\hat{y} \mid x)$. This suggest you should focus your attention on improving the search algorithm.

- True. (X)
- False.

6. Consider the attention model for machine translation. Further, here is the formula for $\alpha^{<t,t'>}$. Which of the following statements about $\alpha^{<t,t'>}$ are true? Check all that apply.

- We expect $\alpha^{<t,t'>}$ to be generally larger for values of $a^{<t'>}$ that are highly relevant to the value the network should output for $y^{<t>}$. (Note the indices in the superscripts.) (X)
- We expect $\alpha^{<t,t'>}$ to be generally larger for values of $a^{<t>}$ that are highly relevant to the value the network should output for $y^{<t'>}$. (Note the indices in the superscripts.)
- $\sum_t \alpha^{<t,t'>} = 1$ (Note the summation is over t.)

- $\sum_{t'} \alpha^{<t,t'>} = 1$ (Note the summation is over t'.) (X)

7. The network learns where to "pay attention" by learning the values $e^{<t,t'>}$, which are computed using a small neural network: We can't replace $s^{<t-1>}$ with $s^{<t>}$ as an input to this neural network. This is because $s^{<t>}$ depends on $\alpha^{<t,t'>}$ which in turn depends on $e^{<t,t'>}$; so at the time we need to evalute this network, we haven't computed $s^{<t>}$ yet.

- True (X)
- False

8. Compared to the encoder-decoder model shown in Question 1 of this quiz (which does not use an attention mechanism), we expect the attention model to have the greatest advantage when:

- The input sequence length T_x is large. (X)
- The input sequence length T_x is small.

9. Under the CTC model, identical repeated characters not separated by the "blank" character (_) are collapsed. Under the CTC model, what does the following string collapse to?
__c_oo_o_kk___b_ooooo__oo__kkk

- cokbok
- cookbook (X)
- cook book
- coookkbooooooookkk

10. In trigger word detection, $x^{<t>}$ is:

- Features of the audio (such as spectrogram features) at time t. (X)
- The t-th input word, represented as either a one-hot vector or a word embedding.
- Whether the trigger word is being said at time t.
- Whether someone has just finished saying the trigger word at time t.