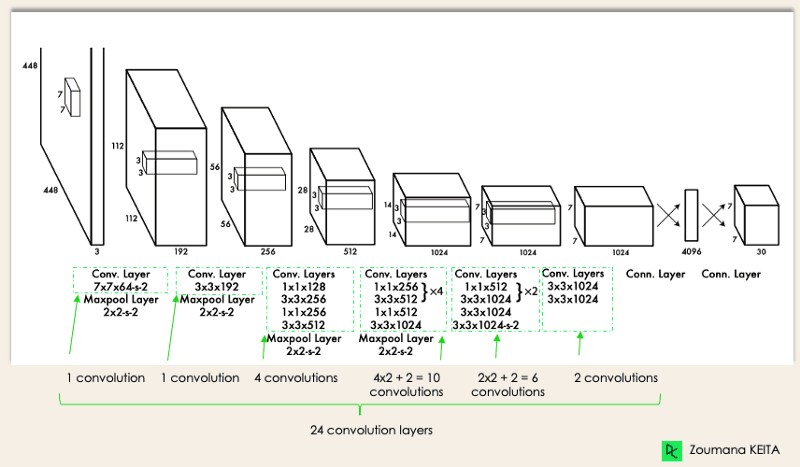
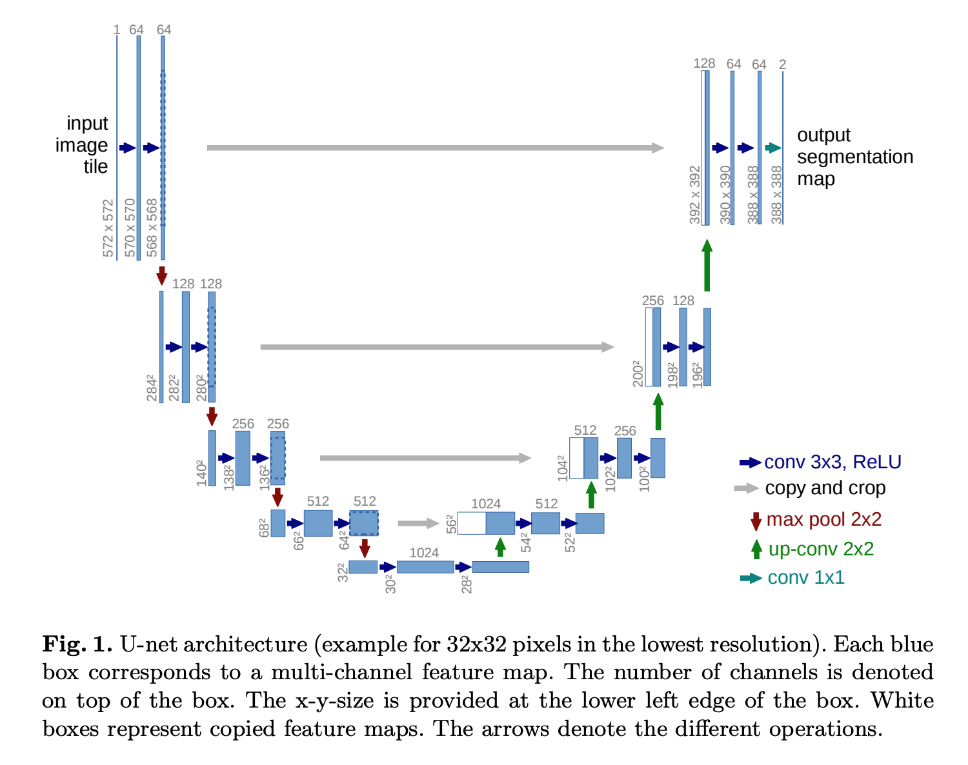
Course Notes:

Conv Nets/ Computer Vision:

1. Week 1
2. Week 3 (Detection Algorithms)
   1. Object Detection (YOLO Algorithm):
      1. Construct target vector for each grid cell (do one for each anchor box shape), perform sliding window
      2. Makes bounding box prediction
      3. For each class run non-max suppression to generate final predictions



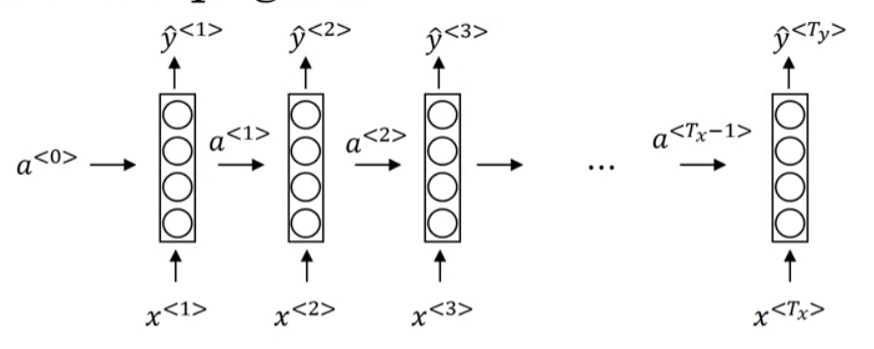
* 1. Semantic Segmentation
     1. U-NET Architecture



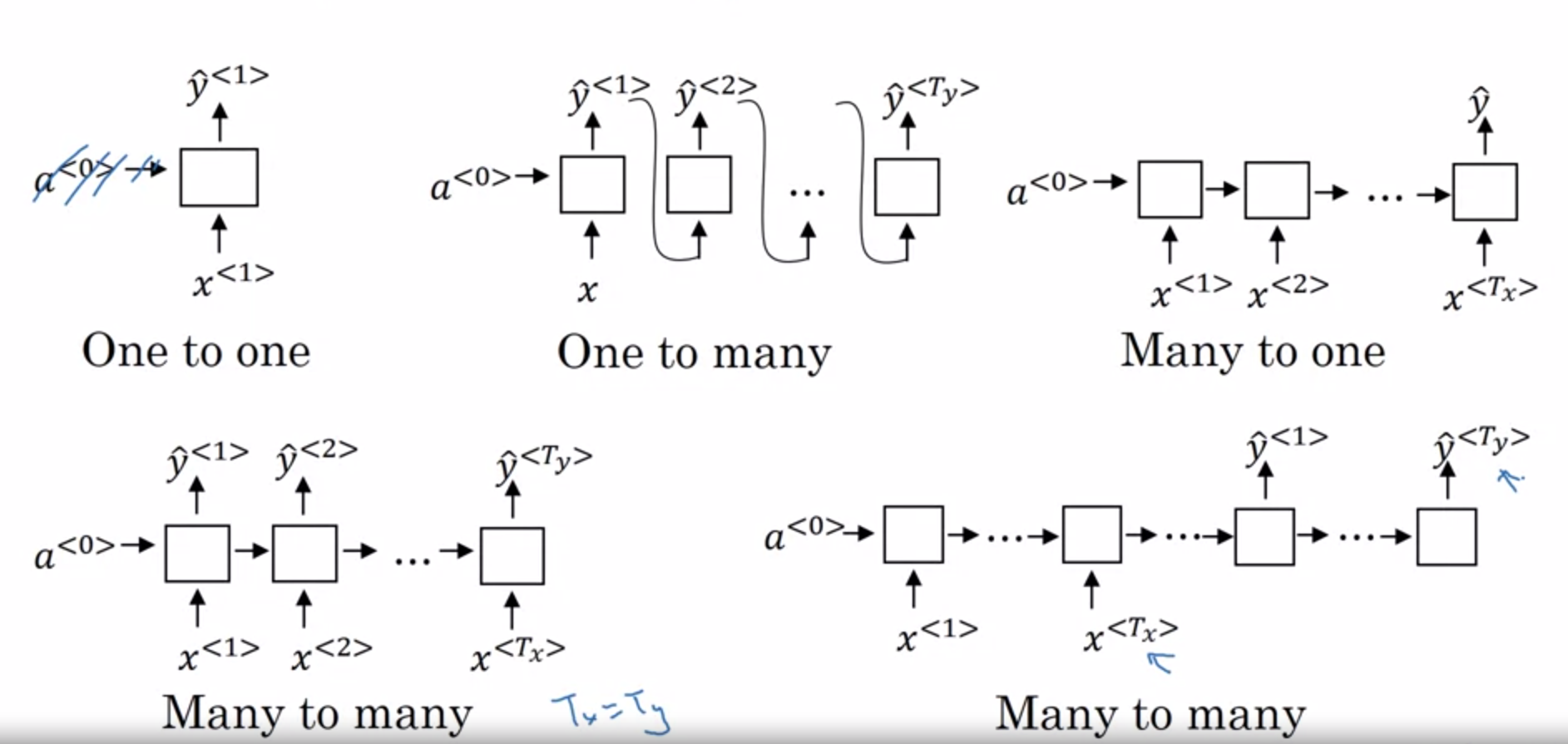
1. Week 4
   1. Face Recognition:
      1. Verification (confirm that image is that of given id)
      2. Recognition (max image with id in database or “not recognized”)
      3. One Shot Learning (need to recognize given just one image)
      4. Learn similarity function: metric on degree of difference between two images
         1. If “same”
         2. Else: “different”
      5. Example: Siamese network
         1. Distance between encoding of images (ex. Norm/norm^2)
      6. Triplet loss function
         1. Build anchor and postive and negative examples (train with triplets)
         2. Need multiple images of each person in training set
         3. Goal: choose triplets tough to train on; i.e. those such that
         4. FaceNet: “[A unified embedding for face recognition and clustering](https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Schroff_FaceNet_A_Unified_2015_CVPR_paper.html)”
         5. State of Art: Datasets of >100 million images; some pretrained models are available
      7. Binary Classification
         1. Adding sigmoid function on dense net combining two encodings
            1. Target label 1 – same, or 0 - different
            2. Can use chi-squared
         2. Can save precomputed encodings to reduce computation
         3. Ref: “[deepFace closing the gap to human level performance](https://openaccess.thecvf.com/content_cvpr_2014/html/Taigman_DeepFace_Closing_the_2014_CVPR_paper.html)”
   2. Neural Transfer
      1. Generate new image in the style of another (Content (C) + Style (S) = Generated (G)
      2. Example: AlexNet (ref. “[Visualizing and understanding convolutional networks](https://link.springer.com/chapter/10.1007/978-3-319-10590-1_53)”):
         1. Find image patches that maximize the unit’s activation over the layers of the network
            1. Layer 1: ex: shade, or edge
            2. Layer 2: composite shapes and patterns (stripes, thin vertical lines, rounded shapes, etc.)
            3. Layer 3: people, wheels, honeycomb
            4. Layer 4: dogs, water
            5. Layer 5: specific objects
      3. Cost Function
         1. ; where is over width, j is over height and k is over channels
         3. Style is the correlation between activations across functions
         4. Procedure:
            1. Initialize G randomly
            2. Use gradient descent to minimize J(G)
            3. Use hidden layer ( to compte content cost hidden layer; generally somewhere in middle, not too shallow or deep
            4. Use unnormalized cross-covariance of a layer for style metric
      4. Style ref: [“A neural algorithm of artistic style”](https://arxiv.org/abs/1508.06576),
   3. Conv nets in 1d and 2d
      1. 1D example: ekg signal
         1. 1d filter allows convolving long the signal
         2. 14x1 \* 5x1 -> 10 x 16 \* 5x16 -> 6x32
         3. Often recurrent networks are used instead on sequence models
      2. 3d data
         1. Example CT scan
         2. Data has height width and depth of scan
         3. H x w x d x c \* 5x5x5x1 (16 filters) -> 10x 10 x10x16 …

Sequence Models

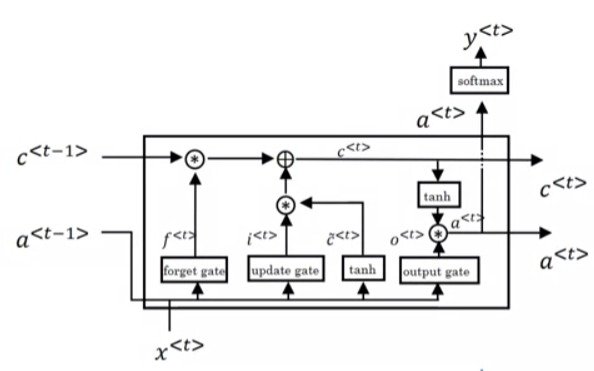
1. Week 1
   1. Examples: speech recognition, music generation, sentiment classification, dna sequence analysis, machine translation, video activity recognition, name entity recognition
   2. Notation:
      1. X: sentence; contains names
         1. one element of sequence (ex. One word)
         2. element of training example
         3. : length of sequence of ith training example
      2. Vocabulary: all words in alphabetical order map each to its nth location in the sequence
      3. Can use one-hot representation for all words
      4. Y: sequence of 1s and 0s corresponding to names and not names for each word
   3. Standard networks don’t tend to work well as
      1. inputs and outputs are often different length
      2. doesn’t share features learned along across different positions of text
   4. Recurrent Neural Network
      1. Inputs activation from prior timestep into next timestep

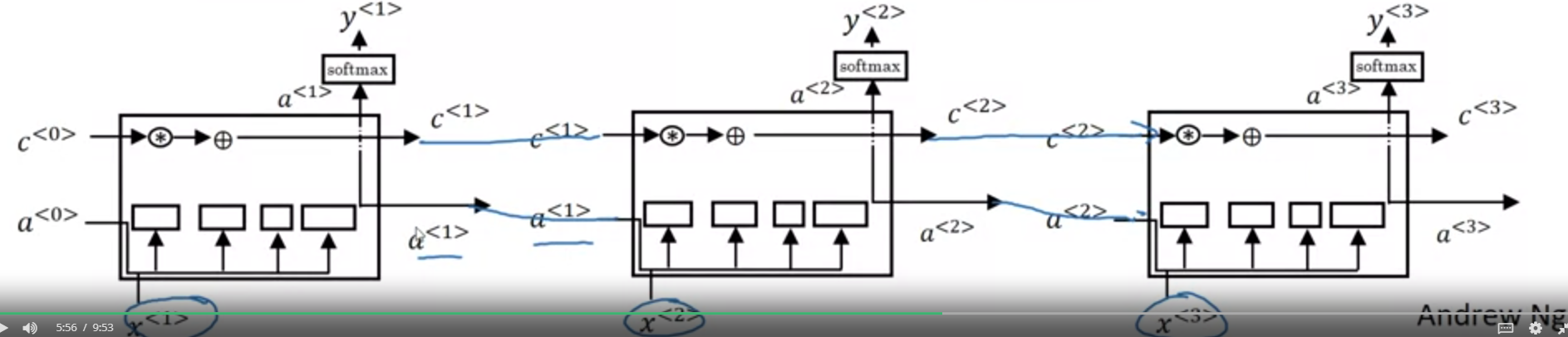


* + 1. Limitation: uses only antecedent information for the task; this is addressed with Bidirectional RNN for tasks where useful
    2. Forward Propagation:
       1. Common activation functions
          1. tanh (sometimes relu) (i.e. )
          2. For output can use sigmoid depending on task (i.e. )
    3. Back propagation
       1. Paramters are
       2. Loss function (Cross entropy loss):
  1. RNN Architectures
     1. Many-to-many (inputs to outputs; ex: word identification)
     2. Many-to-many (Not like sized; Ex. Machine translation)
        1. Encoder - decoder
     3. Many-to-one (Ex. Sentiment Classification; y = 0 or 1)
     4. One-to-many (Ex. Music generation)

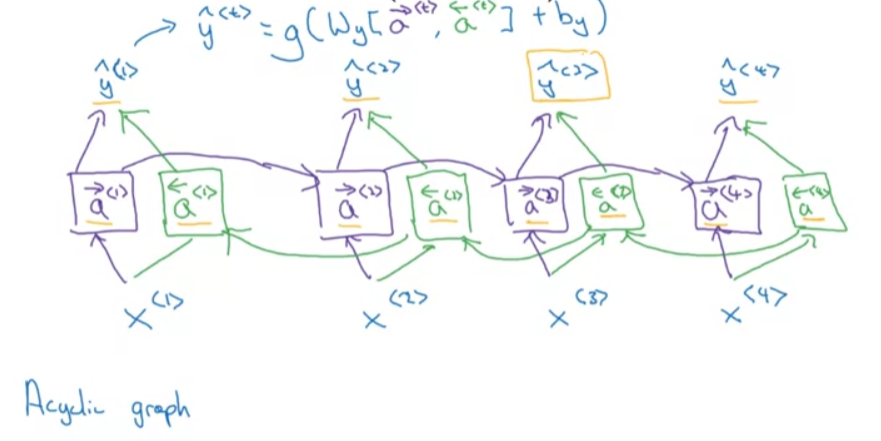


* 1. Language Modelling and Sequence Generation
     1. Speech recognition – converting spoken word to written word
        1. Give probability that it is a certain sentence
        2. Requires large corpus of english text
        3. Tokenize words and end-of-sentence
        4. Predict probability for each with softmax output; get chained conditional probability
     2. Sampling from a sequence
        1. Randomly sample from softmax sequentially
        2. Can also build character-level language model
           1. Pros – never have to deal with unknown words
           2. Cons – much longer sequences and more computationally expensive; harder to optimize
  2. Vanishing gradients with RNNs
     1. Basic RNN is not great with handling long-term dependencies (ex. Singular/plural Subject far from noun)
     2. In general, the gradient for deep networks has a tough time propagating back to the weights of the earlier layers
     3. If see exploding gradients, perform gradient clipping, due to maximum value
  3. Gated Recurrent Unit (GRU)
     1. References
        1. “On the Properties of Neural Machine Translation” <https://arxiv.org/abs/1409.1259>
        2. “Empirical evaluation of gated recurrent neural networks on sequence modeling” <https://arxiv.org/abs/1412.3555>
     2. Helps with vanishing gradient problems; helps capture longer term relationships
     3. Formulation:
        1. c: memory cell; u: update gate () (like a switch); : candidate value; : reset gate
  4. Long Short-Term Memory (LSTM)

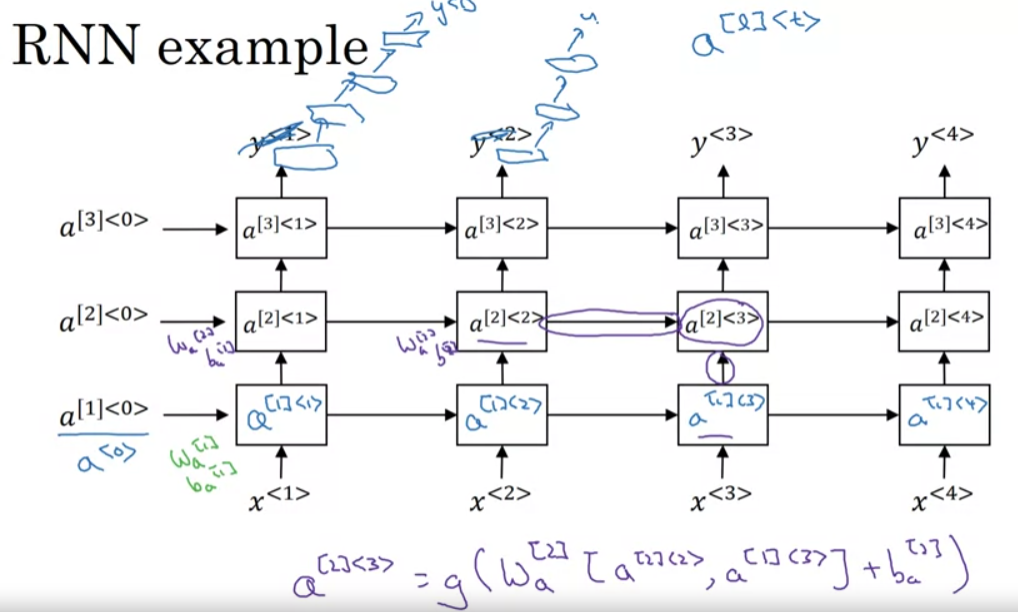


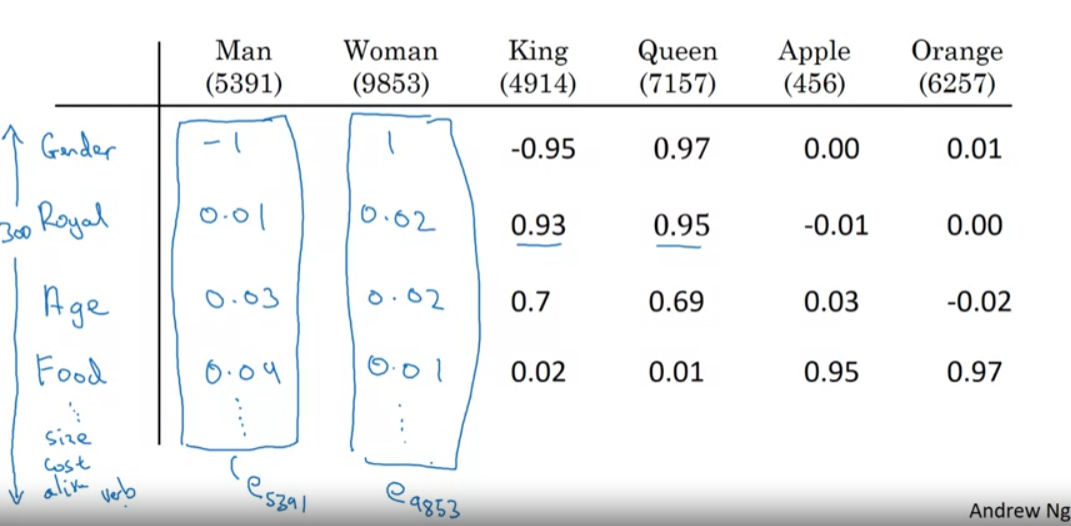
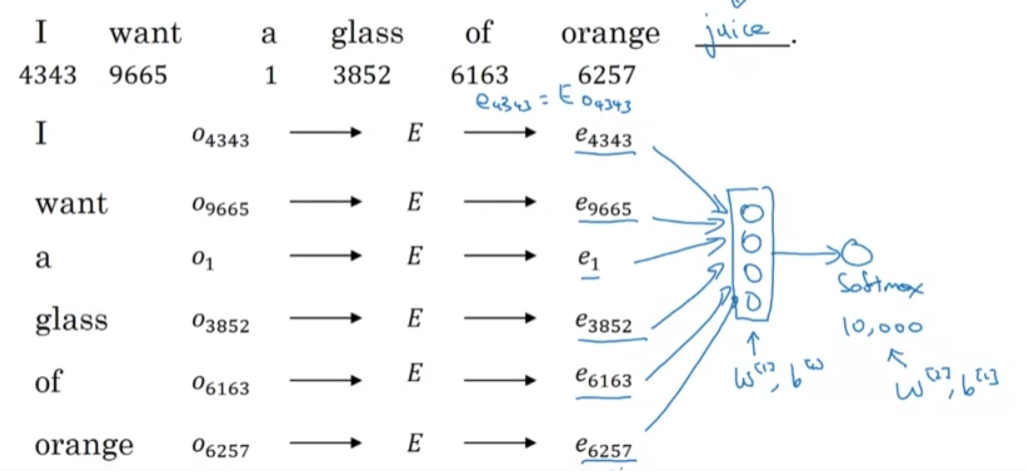
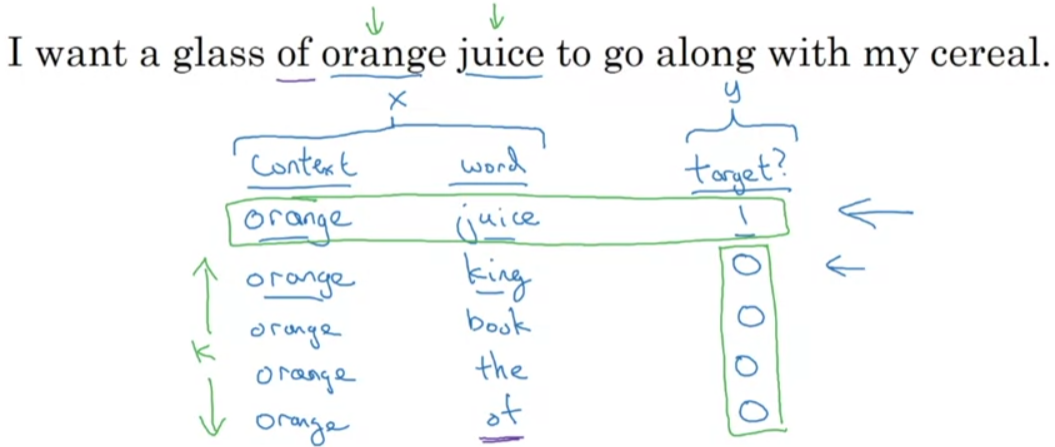
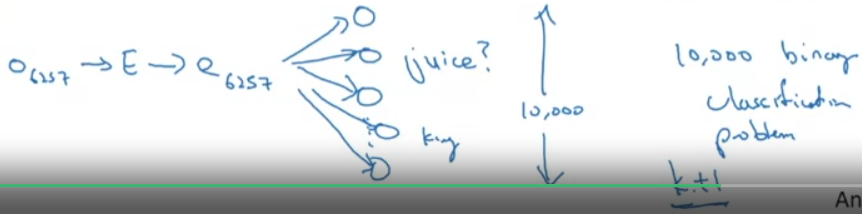
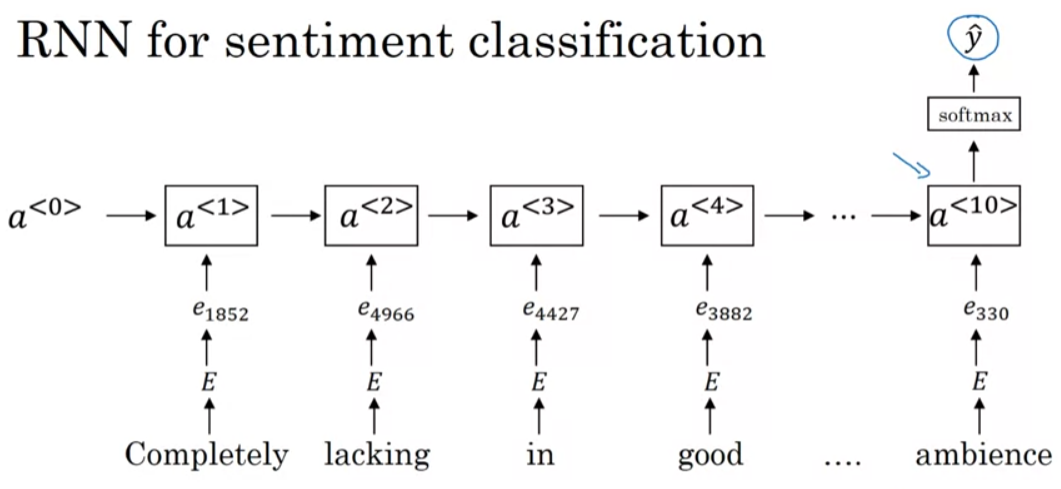


* + 1. References
       1. “Long Short-Term Memory” https://ieeexplore.ieee.org/abstract/document/6795963
    2. Formulation
       1. forget gate, : output gate
    3. For peephole connection
    4. LSTM is more historically proven, but GRUs are getting momentum and they can scale better to bigger problems
  1. Bidirectional RNN



* + 1. Can be standard, but also LSTM or GRUs
    2. Pros: Good for labelling things in a sentence
    3. Cons: Need the whole sequence at the same time
  1. Deep RNNs

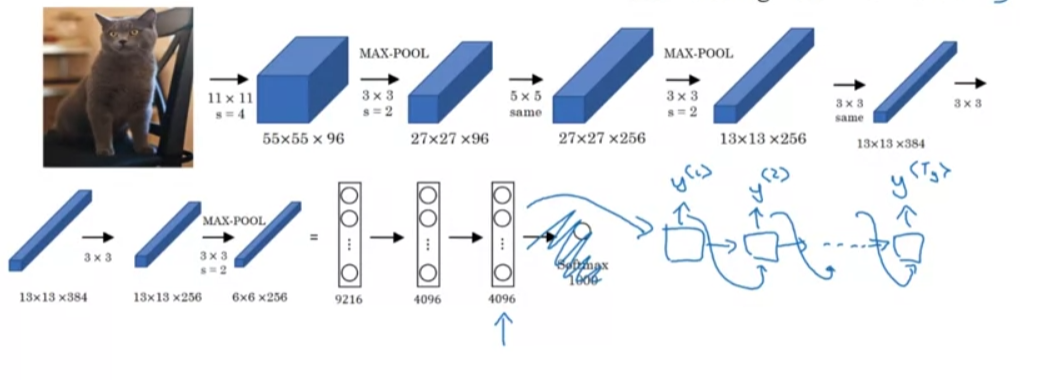


1. Week 2
   1. Word Embeddings
      1. References
         1. Van der Maaten, Laurens, and Geoffrey Hinton. "Visualizing data using t-SNE." *Journal of machine learning research* 9.11 (2008). - <https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf?fbcl>
         2. Mikolov, Tomáš, Wen-tau Yih, and Geoffrey Zweig. "Linguistic regularities in continuous space word representations." *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*. 2013. - <https://aclanthology.org/N13-1090.pdf>
      2. Featurized representation (gives a similarity vector for each feature and then only include a subset of the full language dictionary)
      3. 
      4. Embeddings are often generated by examing billions of words (1-100 B) in unlabelled text; can then transfer to a task with a relatively small training set of words
      5. Optional: continue to finetune the word embeddings with new data
      6. Relationship to face encoding: for face encoding -> siamese network infrastructure; for word embedding have fixed vocab and learns encoding/embedding for each word
      7. Analogies using word vectors: find word that maximizes similarity between one vector difference and another:
      8. Similarity function: cosine similiarity
         1. (most common)
         2. (other option)
      9. Embedding matrix
         1. Learn embedding matrix to turn 10k vocab into 300-length vectors
         3. where
   2. Learning Word Embeddings/ Methodologies for Word Embeddings
      1. References:
         1. Bengio, Yoshua, Réjean Ducharme, and Pascal Vincent. "A neural probabilistic language model." *Advances in neural information processing systems* 13 (2000). - <https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf>
         2. Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013). - <https://arxiv.org/pdf/1301.3781.pdf%C3%AC%E2%80%94%20%C3%AC%E2%80%9E%C5%93>
         3. Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014. - <https://aclanthology.org/D14-1162.pdf>
         4. Bolukbasi, Tolga, et al. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." *Advances in neural information processing systems* 29 (2016). - <https://proceedings.neurips.cc/paper_files/paper/2016/file/a486cd07e4ac3d270571622f4f316ec5-Paper.pdf>
      2. Neural network model to create embeddings:
         1. 
         2. Build word embedding through target word prediction (in the example: juice)
      3. Other context/target pairs
         1. Given four words to left and four words to right to predict word in the middle and use that to construct embeddings
         2. Last 1 word
         3. Nearby 1 word (skip-gram)
      4. Word2Vec (skip-grams)
         1. Set up supervised learner to predict a randomly chosen word +/- 10 word window given a context word
         2. Model:
            1. vocab size: 10k
            2. context
            3. parameter associated with output
            4. Loss:
         3. Problems with softmax classification
            1. As vocab size gets large, calculating the denominator sum gets costly
            2. Option: Hierarchical softmax (create tree of binary classifiers) -> cost scales with log(|v|), where |v| is the size of the vocab
         4. How to sample the context
            1. Option 1: uniformly sample -> get articles/ prepositions super frequently
            2. Option 2: Distribution is not taken randomly and heuristics to bias (better for uncommon words)
      5. Negative Sampling
         1. Sample context and target word
         2. 
         3. Procedure:
            1. Pick context word from target word
            2. Pick a target word from sampling window (positive target)
            3. Pick a set (k) of random words and label as 0 (negative targets)
            4. Create supervised learning model and try to predict target
         4. Model as essentially logistric regression model
            1. 
         5. How are negative targets sampled
            1. Empirically what works best: sample proportional to frequency of word usage in corpus to ¾ power
      6. GloVe Word Vectors
         1. Not used quite as often as word2vec models
         2. “Global vectors for word representation”
         3. # of times appears in the context of within corpus
         4. Minimize
         5. 0 if =0
         6. enables weighting for liklihood/commonality of pairing
         7. Convention: 0log0 = 0
      7. Note on featurization view of word embeddings: You cannot guarantee that first axis of embedding vector is aligned with interpretable axis
   3. Applications:
      1. Sentiment Classification problem
         1. Mapping text review/comment to numerical mapping (i.e. star ratings)
         2. 10k-20k word training set might not be uncommon
         3. Can embed each word and take knowledge from general corpus and apply to problem
         4. Average embeddings and pass to softmax for 5 possible outcomes (1-5 stars)
         5. Problem: average individual words can miss group negations (ex: “completely lacking in … good ambience, good service, etc.” -> has positive rating)
         6. Feed embedding vectors into RNN to predict (many-to-one architecture)
         7. 
      2. Debiasing Word Embeddings
         1. Bias: gender, ethnicity, sexual orientation biases
         2. Learned embeddings can reflect gender, ethnicity, age, sexual orientation, and other biases of the text used to train the model
         3. Example
            1. Say we’ve learned an embedding
            2. Identify bias direction (here let’s focus on gender)
            3. Take embedding vector for he and subtract for she (1a)
            4. Take average or SVD of vectors (1b)
            5. Neutralize: for ever word that is not definitional (i.e. grandmother vs grandfather) project to get rid of bias (2)
            6. Equalize pairs (3)
         4. How do we decide what to neutralize? (doctor vs beard)
            1. Train a classifier, most words are not definitonal (relatively small subset that are)
            2. Full algorithm is a bit more complicated
2. Week 3: Sequence Models and Attention Mechanism
   1. Sequence to Sequence Models
      1. Examples:
         1. Machine translation (translating from french to english)

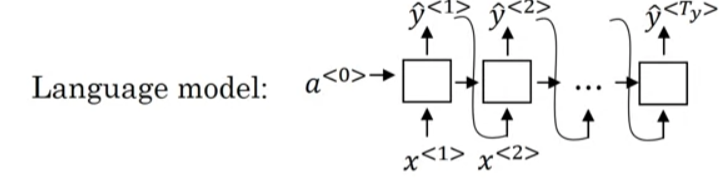


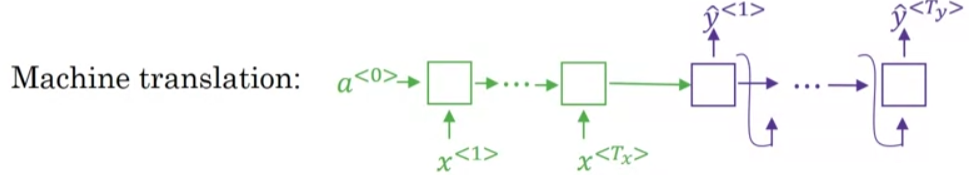
* + - * 1. “Sequence to sequence learning with neural networks” https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html
        2. “Learning phrase representations using RNN encoder-decorder for statistical machine translation” - https://arxiv.org/abs/1406.1078

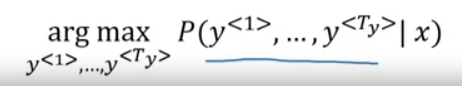
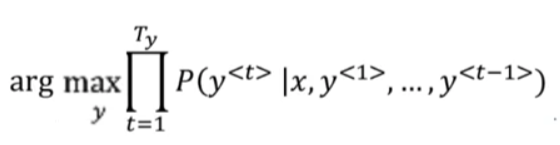
1. image captioning
   1. Architecture: AlexNet -> RNN



* 1. References:
     1. “Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN)” - <https://arxiv.org/abs/1412.6632>
     2. “Show and Tell: A Neural Image Caption Generator”- https://www.cv-foundation.org/openaccess/content\_cvpr\_2015/html/Vinyals\_Show\_and\_Tell\_2015\_CVPR\_paper.html
     3. Picking the most likely sequence
        1. Language model: probabilistic model
        2. Machine translation: conditional language model





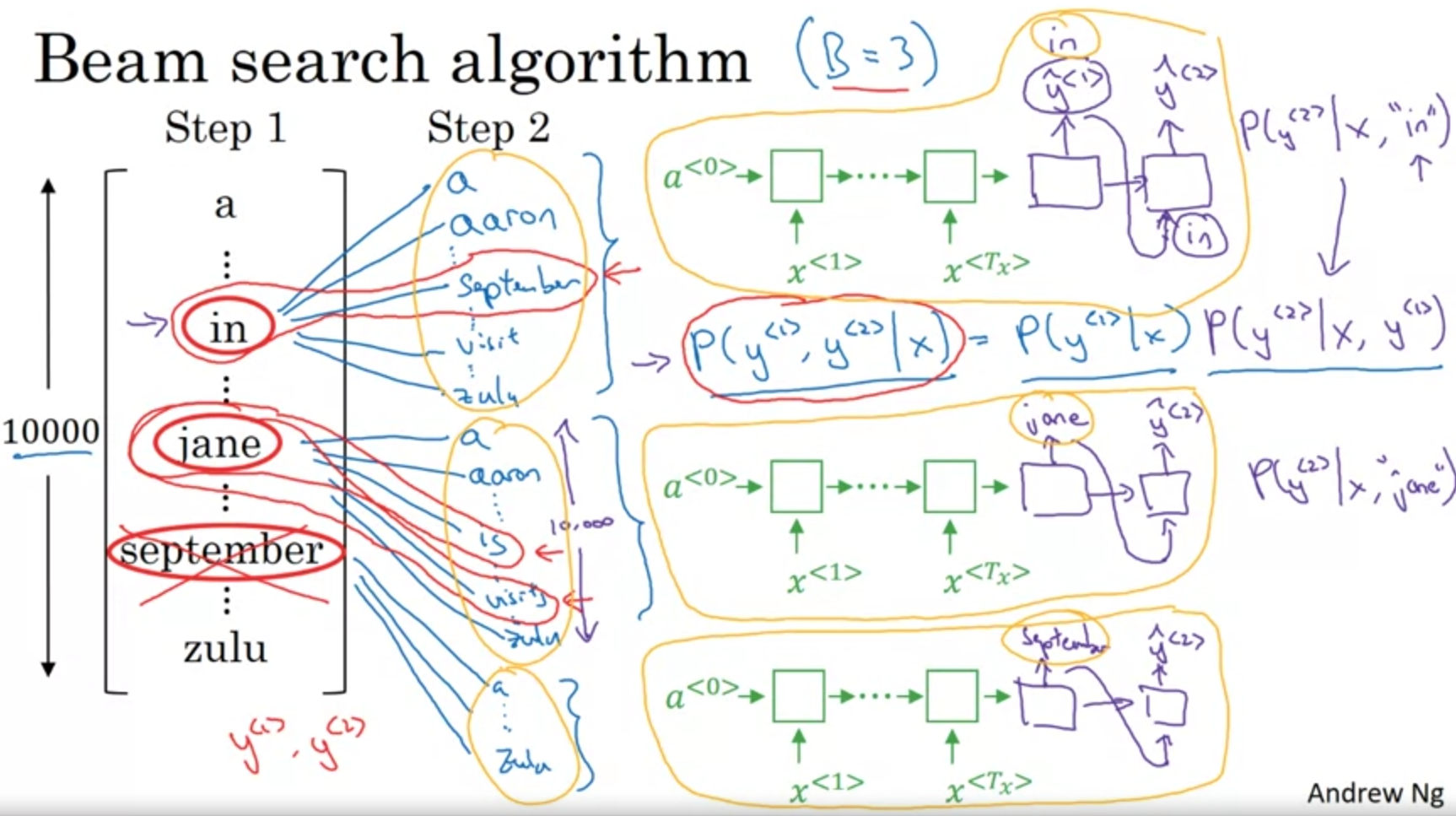
* + - 1. Would like to find: 
      2. Greedy search:
         1. Def: Pick the likely first word from language model and pick the next word that seems most likely
         2. Doesn’t tend to work very well
      3. Need Approximate search algorithm -> Beam Search
         1. Goal: 
         2. Algorithm

Set beam width (B)

Try to find probability of first word given x and keeps top B most probably words

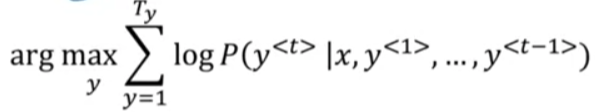
For each B choices pick the top B pairs of all the B\*len(vocabulary) choices

Continue on each until you have B choices that have reached <EOS>

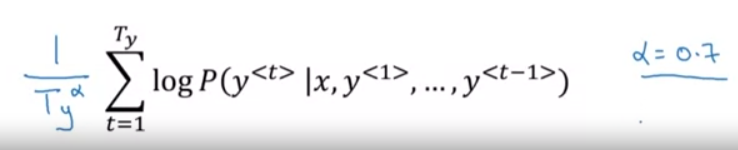


* + - * 1. Note: at each step instantiate “B” copies of the network
        2. Refinement:

Length normalization

In practice, for greater computational stability: 

Due to construction, this loss function favors shorter sentences

Solution (This is length normalization; is tunable, and a hack solution that has shown to be fruitful): 

* + - * 1. B selection tradeoffs

Large B: better result, slower; when very large returns tend to be diminishing

Small B: worse results, faster

General sizes

Production: 10-100

Research: 1000-3000

* + - * 1. Error Analysis

Using both RNN (encoder-decoder) and Beam Search, would be helpful to ascertain the error due to each

Compute and

Case 1:

Beam search chose , but attains higher ; i.e.

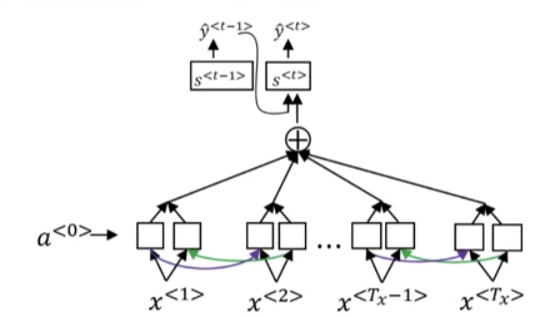
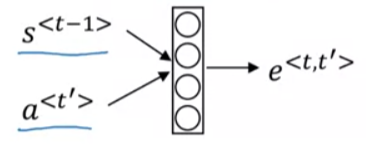
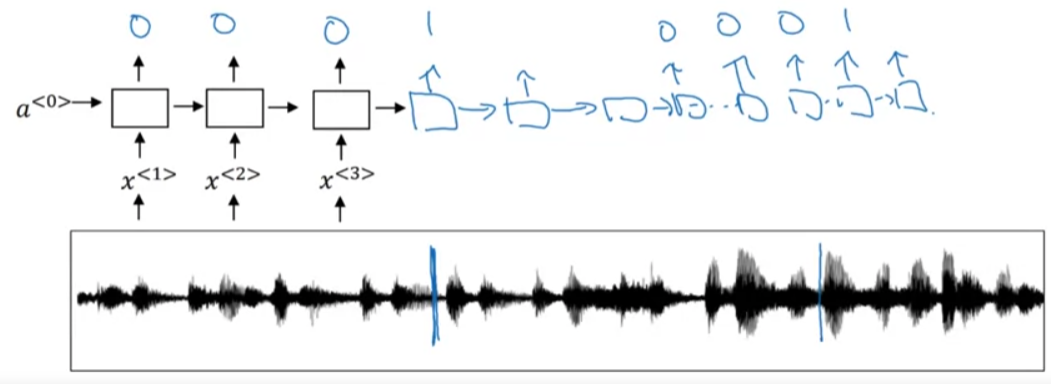
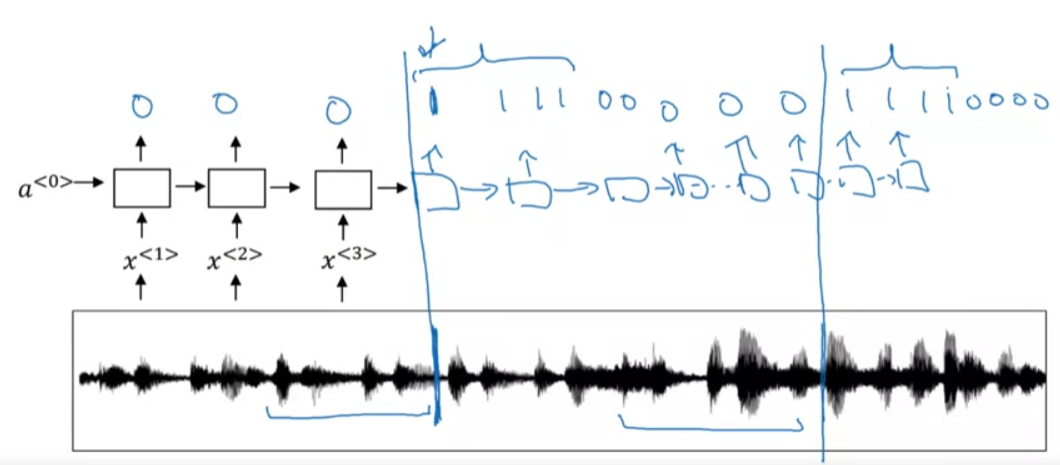
Conclusion: beam search is at fault

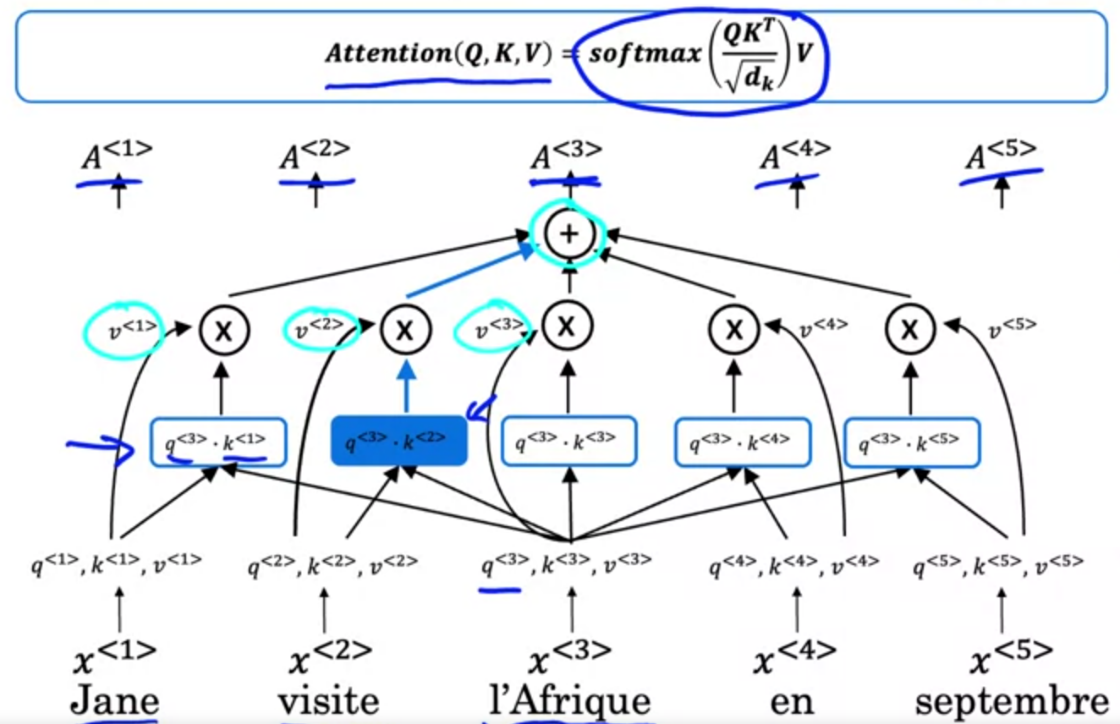
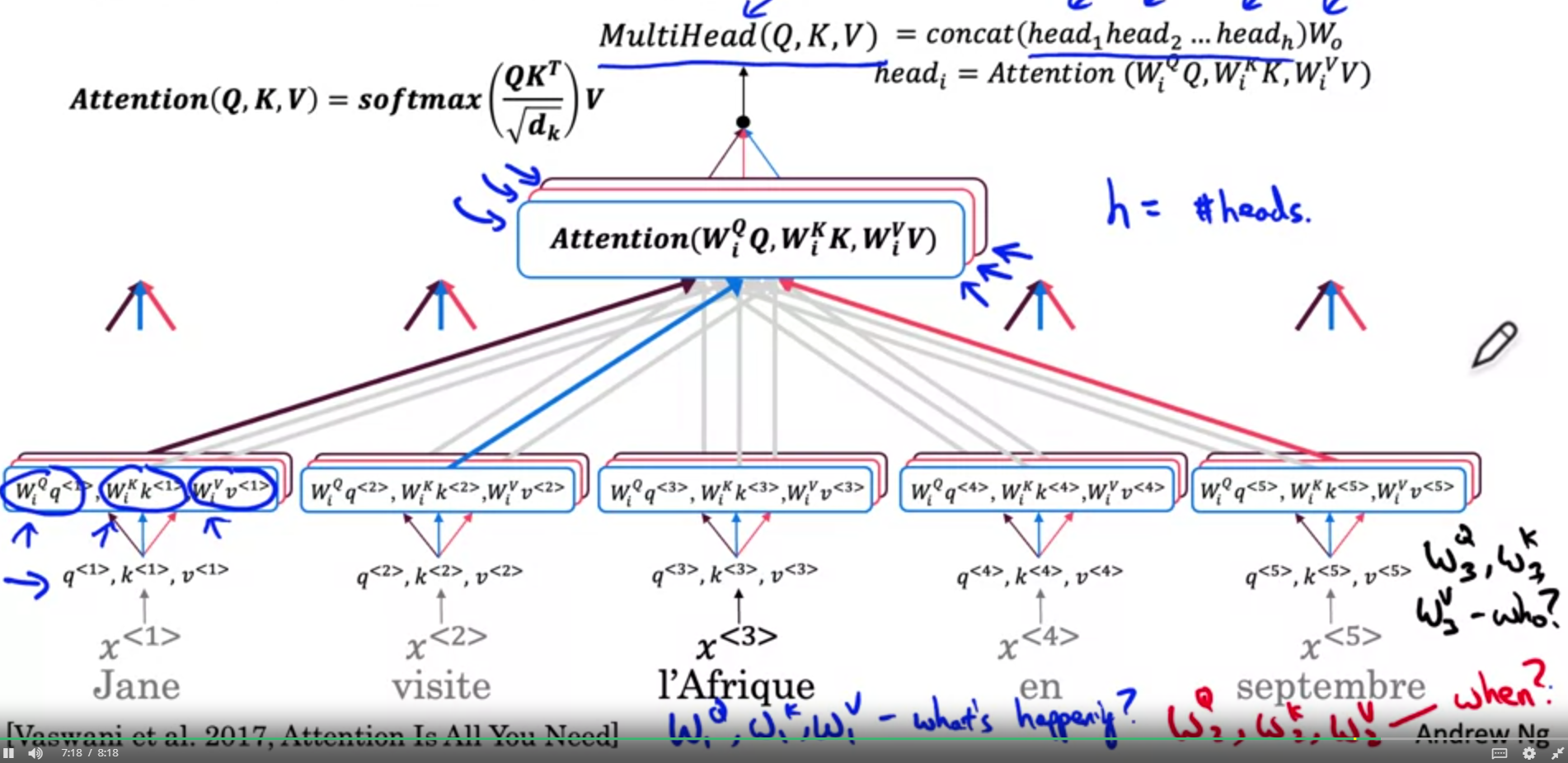
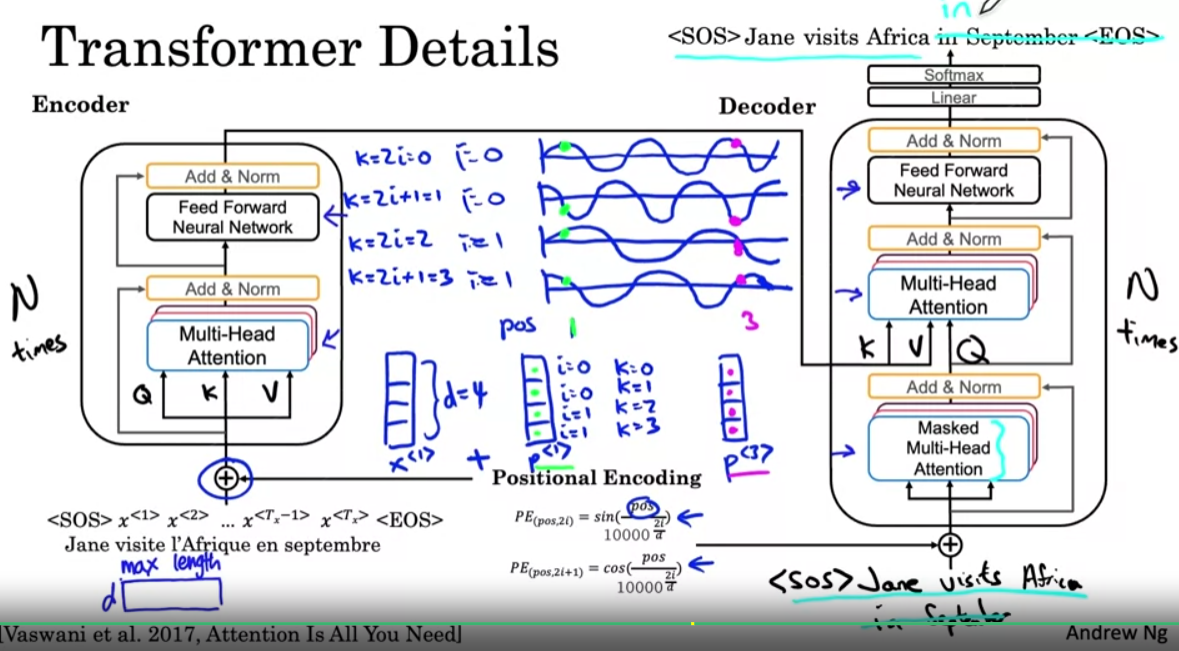
Case 2

Conclusion: RNN model is at fault

Note: If using length normalization: should evaluate the optimization objective for length normalization

Task: Run analysis for each poor output by algorithm and identify which ffraction are due to beam search vs. RNN model

* 1. Attention Models
     1. References:
        1. “Neural machine translation by jointly learning to align and transfer” - <https://arxiv.org/abs/1409.0473>
        2. “Show attention and tell: neural image caption generation with visual attention” - <https://proceedings.mlr.press/v37/xuc15.html>
     2. Modification of encoder-decoder network
     3. Attention model computes a set of weights to pay greater attention to a set of inputs
     4.  
     5. amount of attention should pay to
     6. Train neural network and use optimization algo (i.e. gradient descent) to learn weights
     7. Con: takes quadratic cost (
  2. Speech Recognition
     1. Given audio clip, generate transcript
     2. Commonly preprocessed by transforming audio clip into spectogram
     3. References:
        1. “Connectionist Temporal Classification: Labeling unsegmented sequence data with recurrent neural networks” - <https://dl.acm.org/doi/abs/10.1145/1143844.1143891>
     4. Connectionist Temporal Classification:
        1. Collapse repeated characters: “ttt\_h\_eee\_\_\_\_\_ \_\_\_\_\_qqq” => “the q”
     5. Trigger word detection
        1. First approach:
           1. 
           2. Leads to imbalanced training set -> set to range instead of instance instead
           3. 
        2. Words that activate; Ex. “alexa”, “ok google”, “hey siri”
        3. Take audioclip -> make spectrogram features -> define target label

1. Week 4: Transformers
   1. Reference:
      1. “Attention is All You Need” <https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>
   2. RNN -> GRU -> LSTM (Sequential Models)
      1. Increased complexity
      2. each unit is a bottleneck due to the fact that it is a sequential model
   3. Transformer: Attention + CNN
      1. Self-attention: ; calculate attentions for each in parallel
      2. Multi-head attention: loop over self-attention process
      3. Parallel processing like a CNN
   4. Self-attention mechanism
      1. 
      2. = attention-based vector representation of a word
         1. : query, : key, : value
         3. Compute softmax of inner products of q,k and multiply through with value and sum calues
      3. For each element in the sequence (word), have query key, value;
         1. query enables asking question about element
         2. the key looks at all other elements and based on similarity to those elements is the most relevant to that question
         3. the value allows the representation to plug in how the relevant element should be represented in the context of the original element
         4. This enables an element to be modified by surrounding elements
         5. i.e.
            1. Q = interesting questions about the words in a sentence
            2. K = qualities of words given a Q
            3. V = specific representations of words given a Q
   5. Multi-headed attention
      1. 
      2. Calculate attention for each multiple times enabling greater set of optionality of the questions being asked via the queries
      3. These heads are calculated in parallel and concatenated together
      4. – corresponds to the index of the head
   6. Transformer Network
      1. 
      2. Encoder Block (repeated N times; typical N=6)
         1. multi-head attention layer (Q,K,V)
         2. passed into feed forward neural network
      3. Decoder Block (repeated times)
         1. Input first few words (for NLP sentence <SOS>)
         2. Multi-head attention block 1 -> multi-head attention block 2 with values from encoder
         3. Run sequentially
      4. Additional Bells and whistles
         1. Positional encoding; polar conversion
            1. – dimension of word embedding ()
            2. Generates unique positional encoding value
         2. Masked Multi-head attention
            1. Used in training process; when training have full correct output and therefore can repeatedly “pretend” that it has perfectly predicted the first few elements and check if it can predict the remainder
      5. Outputs of multi-head attention layers and feed forward neural network share the same shape (d x max length)
         1. Max length – maximum length
      6. Positional encoding is also passed through residuals
      7. Add & Norm layer
         1. Plays similar role to batch normalization
      8. Decoder outputs into linear layer with softmax activation