Language

Arthur J. Redfern
arthur.redfern@utdallas.edu

Outline

- Motivation
- Character and word embeddings
- Language modeling
- Sentence embeddings
- Conditional modeling
- Translation
- Text classification
- Summarization
- Part of speech tagging
- Question answering
- Dialogue agents
- Image captioning
- References

2 nice sites with links to state of the art techniques for many language topics

Tracking progress in natural language processing

- http://ruder.io/tracking-progress-nlp/
- https://github.com/sebastianruder/NLP-progress

Motivation

Why This Abbreviated Choice Of Topics

- Embeddings allow characters, words and larger units to be worked with conveniently from a mathematical perspective and are needed for most all language processing tasks
  - It's a foundational component that affects the accuracy of all subsequent language processing
- Language modeling is used in many places including language translation and speech to text transduction
  - Since it comes up in so many applications it's worthwhile to look at it in some detail
  - As with embeddings, efficient language modeling impacts all downstream language applications
  - A low of attention is being focused on this right now because of this (no pun intended)
- Language translation builds on sequence to sequence models and is an excellent success story within deep learning
  - Seeing similar models in different settings helps with the understanding
  - Language translation can be used with speech applications to build larger composite applications

An Observation

- There's a set of language related tasks that are similar to vision, speech, ... tasks
  - Ex: Translating a sentence in 1 language to a sentence in another language (sequence to sequence)
  - Ex: Classifying a review as positive or negative (classification)
  - Ex: Generating text (generation)
  - ...
- There's a part of language that feels a little different
  - Understanding what text means
  - This feels more fundamental to AI
  - But I'm not sure if it is or not, we'll have to see how things play out

Disclaimer

- There's a lot of language related stuff not included here
    - Different methods within the categories of problems included here
    - Problems that are not included here
- Possibly some of this will be addressed in future versions of the slides
- Regardless of whether it is or not, hopefully these slides provide enough of a base from which to branch off and learn more on your own

Character And Word Embeddings

Character Embedding

- From a mathematical perspective it’s cumbersome to directly work with characters
- So instead assign a vector to each character (embed a character into a vector)
- For languages with a small number of characters, a 1 hot encoding is commonly used
  - We’ve already seen this in the speech slides
- Example (additional elements can be added for numbers, punctuation, ... depending on the application of interest)
  - A = [1, 0, 0, ..., 0, 0]
  - B = [0, 1, 0, ..., 0, 0]
  - ...
  - Z = [0, 0, 0, ..., 0, 1]

Word Embedding

- From a mathematical perspective it's cumbersome to directly work with words
- So instead assign a vector to each word or word piece (embed a word or word piece into a vector)
  - Not practical to use a 1 hot encoding because there are so many words and the vector would be too long (~ 13M for English)
  - Instead use a length 100 – 1000 vector of dense real values (discuss: why not use just a single real? hint: number of neighbors)
  - Past some length there are diminishing returns in accuracy improvements in the applications that use them
- It's useful if the word to vector assignment exhibits some language understanding
  - E.g., the cosine of the angle between vectors corresponding to similar words should be close to 1
- Why?
  - Subsequent processing on the vectors is typically going to do feature extraction, classification and generation
  - Consider feature extraction: it makes sense that it's easier to extract features if the feature extractor only needs to learn meaning (or whatever the task) vs learning both meaning and a random mapping
  - Consider generation: it makes sense that if the generator is off by a little and generates a synonym to the target word vs a word with no relationship

The Distributional Hypothesis

- The methods used to embed words in dense vectors in this section all rely on the distributional hypothesis
  - Hypothesis: words that are used in the same context tend to have similar meanings
  - A word is described by the company it keeps (perhaps also applicable to humans)
  - Per the previous slide this meaning should be accounted for in the assignment of vectors
- This leads to a typical set of parameters, the choice of which helps determine the assignment
  - Context type and window
  - Frequency weighting, dimensionality reduction and similarity measure
- The choice of the embedding determines how easy or difficult it is for the extrinsic / downstream task to extract the information that it needs from the embedding

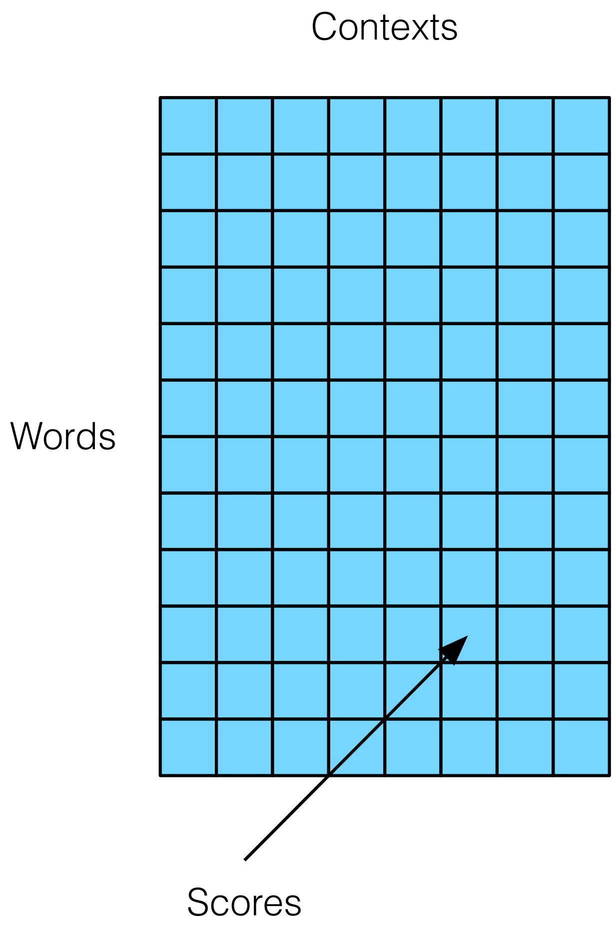
List from https://en.wikipedia.org/wiki/Distributional_semantics

History

- For a nice history of word embeddings see
  - An overview of word embeddings and their connection to distributional semantic models
  - http://blog.aylien.com/overview-word-embeddings-history-word2vec-cbow-glove/
- 2 basic categories
  - Count based methods
  - Predictive methods

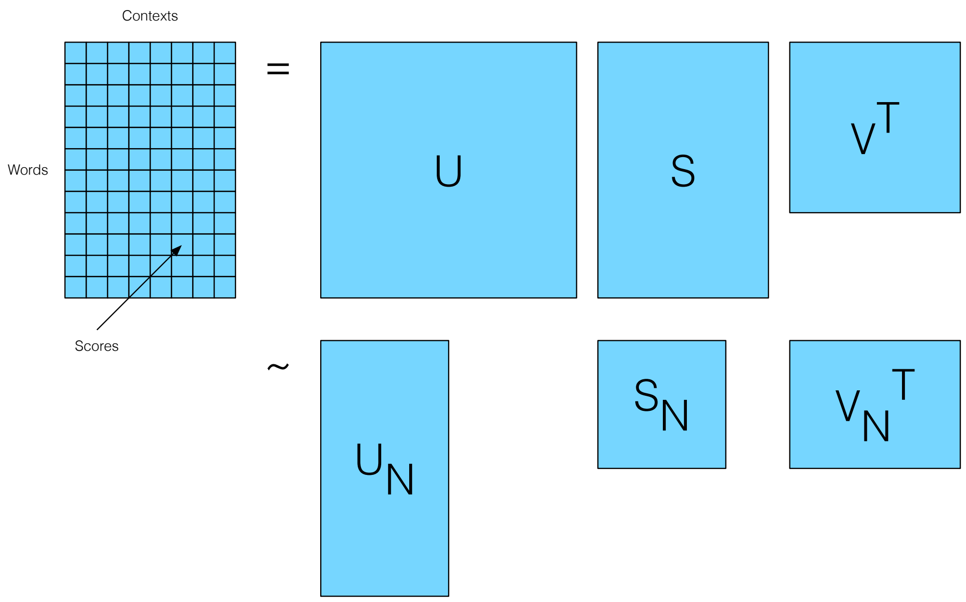
SVD Based Word Embedding

- Idea
  - Use the context of words around a word to describe a word
  - Ideally have a large text with billions of phrases and sentences to learn this from
- Parse the text into pairs
  - Words (W unique)
  - Contexts (C unique)
    - Ex: context can be the proceeding word (bigram)
    - Ex: context can be the whole document (latent semantic analysis)
- Form a matrix X
  - Unique words as rows
  - Unique contexts as cols
  - Entries are counts, probabilities, normalized probabilities, positive point wise mutual information (a normalized measure of the word – context pair likelihood); sometimes these are raised to a power like 3/4 to introduce some smoothing



SVD Based Word Embedding

- Take the SVD of X and keep the cols of U, rows and cols of Σ and rows of V corresponding to the N largest singular values
  - $X = U \Sigma V^T \approx U_N \Sigma_N V_N^T$



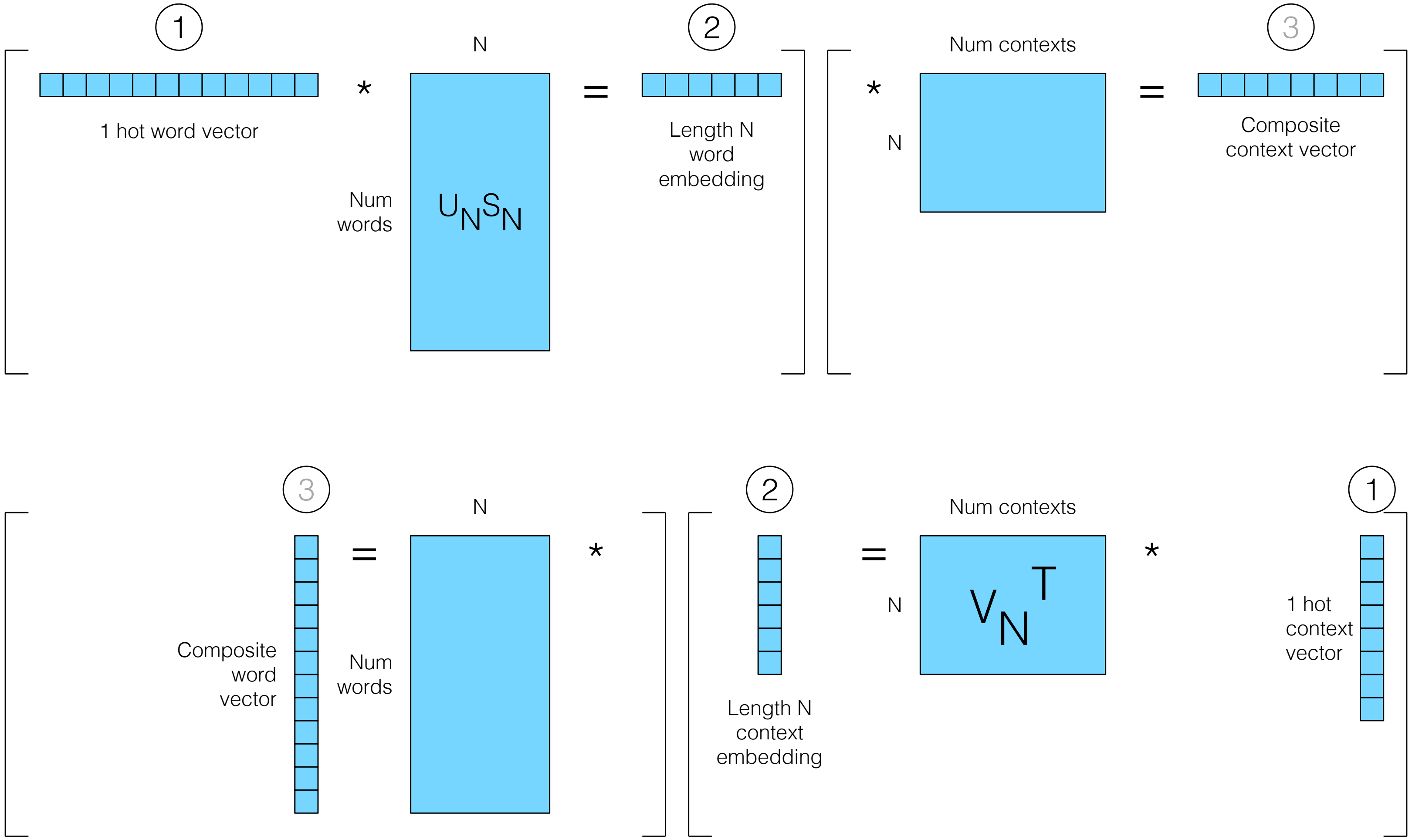
- Embedding
  - Each row of $U_N \Sigma_N$ is a length N embedding for the corresponding word
  - Each col of V_N^T is a length N embedding for the corresponding context

- Comments
  - The linear map from 1 hot input word vector to length N word embedding can implement all mappings
    - Just make the row the desired mapping for each word
    - So the question really is what it the best linear mapping?
  - The linear mapping is a result of 2 choices
    - The matrix representing words and context likelihoods
    - The choice of the factorization method applied to this matrix to create word mapping and context mapping matrices

Note the asymmetry of the word vs context embedding; sometimes this is modified such that Σ_N is distributed partially to both the word and context embedding as Σ_N^p and Σ_N^{1-p}

SVD Based Word Embedding

Step 3 is not part of the embedding, just interesting to think about



Word2Vec

- Word to vector mapping strategy
  - Assign a vector to a word that's a function of the words likely to be around it
  - Don't have to explicitly know the meaning of the word just "the company that it keeps”
  - Learn an embedding matrix and a bias (which could later be combined into the embedding matrix)
- 2 methods for creating the mapping
  - Continuous bag of words
  - Skip gram (more popular)

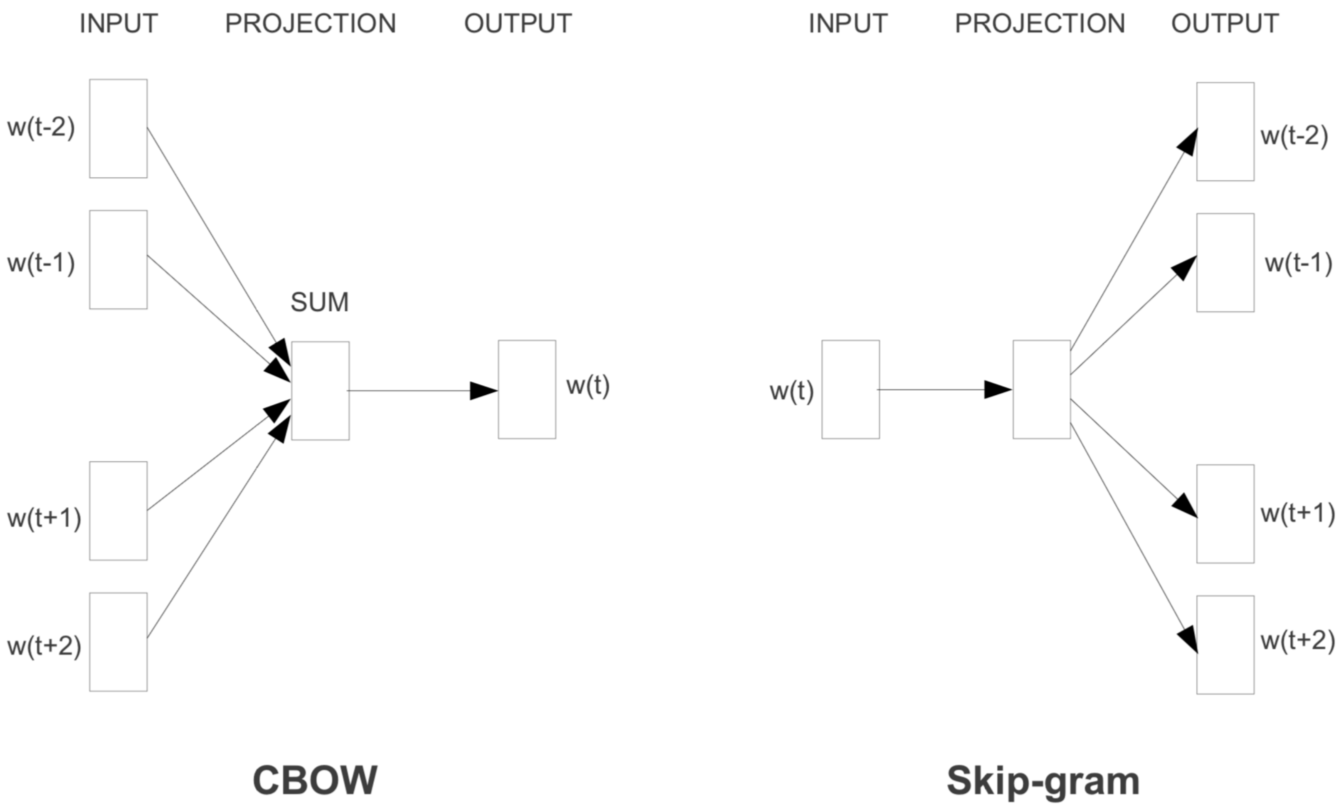
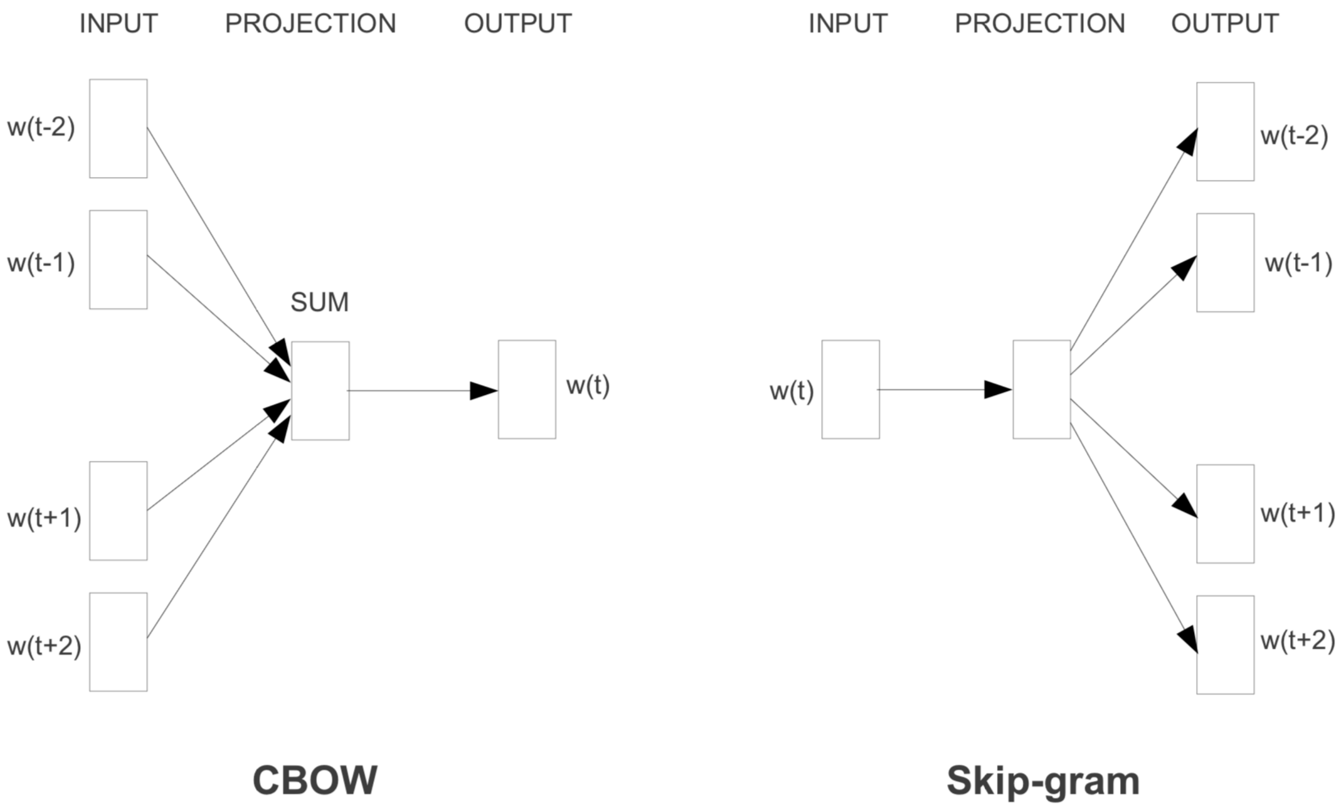


Figure from https://arxiv.org/abs/1301.3781

Word2Vec

- Continuous bag of words
  - Model predicts the current word from surrounding context of words
  - Ordering of context words is not considered
  - Slightly better with small data sets
  - Slightly worse with large data sets
  - Context window size of ~ 5
- 2 layer standard neural network
  - Input: vector with 1s in 1 hot word locations
  - 1: embedding length x num words NN layer
  - 2: num words x embedding length NN layer
  - Output: vector with 1 in 1 hot word location
- Objective
  - Training minimizes the negative log likelihood of the current word given the context
  - $J = -\log p(w_t \mid w_{t+n}, \ldots, w_{t+1}, w_{t-1}, \ldots, w_{t-n})$

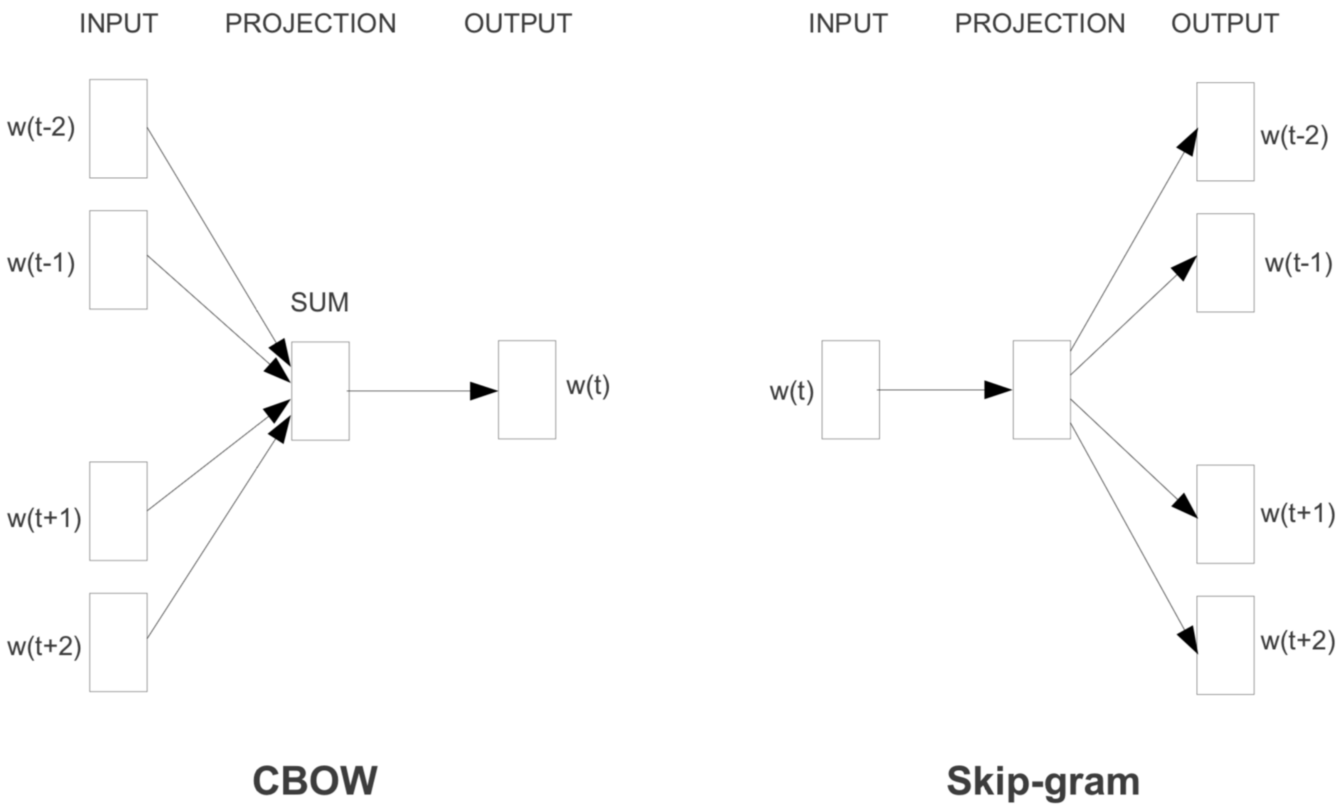


After training, the linear transformation from layer 1 is used for the word embedding for both the CBOW and skip-gram models

Figure from https://arxiv.org/abs/1301.3781

Word2Vec

- Skip gram
  - Model predicts surrounding context of words from the current word
  - Near words are weighted more heavily than far words based on context window range selection during training
  - Does a better job with infrequent words, slightly worse with small data sets, slightly better with large data sets
  - Maximum context window size of ~ 10
- Same 2 layer standard NN structure
  - Just with a single 1 hot encoded input
- Objective
  - Training minimizes the negative log likelihood of the context given the current word
  - $J = -\sum_{-n \le j \le n, \ne 0} \log p(w_{t+j} \mid w_t)$



Using 1 input word to predict multiple context words feels imbalanced from a mapping perspective (it is); however, this is only needed during training and it's handled by turning the 1 to many grouping into multiple 1 to 1 pairs / training samples (potentially each for use with more negative pairs depending on the specific training strategy selected)

Figure from https://arxiv.org/abs/1301.3781

Word2Vec

- Various methods for reducing training complexity are used
  - Hierarchical softmax
  - Negative sampling (more common); this only updates the network corresponding to a few (e.g., 2 – 20) of the negative elements in the output vector each sample instead of updating all of them all of them
- Various methods for improving accuracy are used
  - Sub sampling high frequency words by increasing the likelihood of skipping a training sample based on the frequency of it's occurrence (e.g., skip "the" training samples a lot)
- It turns out that the training method has a relatively large impact on performance

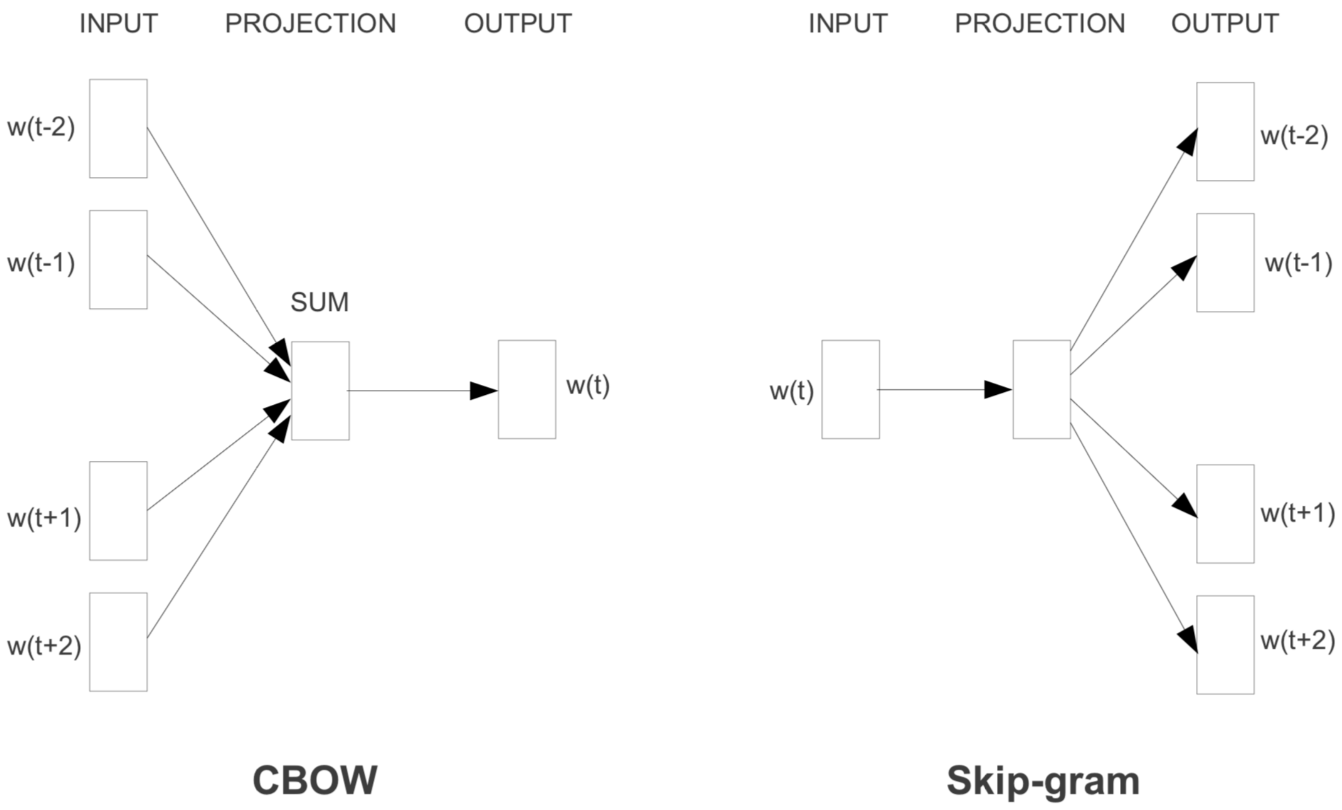


Figure from https://arxiv.org/abs/1301.3781

Word2Vec

Simple vector addition on the resulting word embeddings leads to many reasonable results (implying some basic language understanding)

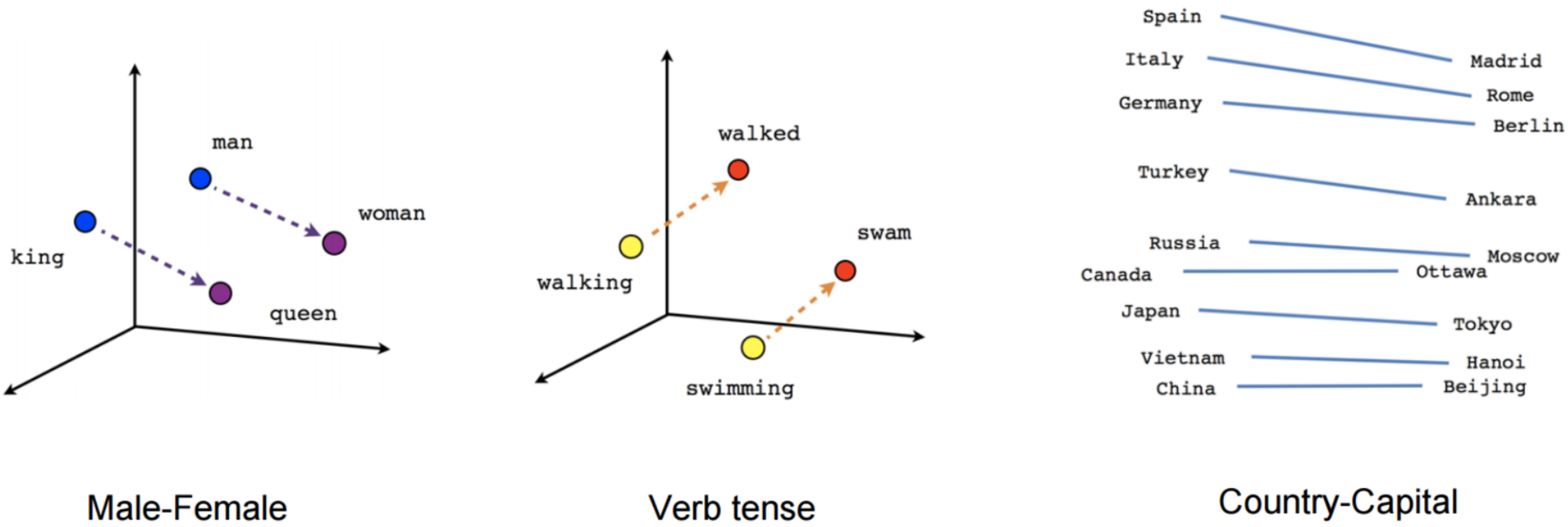


Figure from https://thegradient.pub/nlp-imagenet/

Glove

- Consider the 2 approaches we've seen so far
  - SVD based methods use full data statistics but tend to do poorly on word analogies
  - CBOW and skip gram methods do well on word analogies but poorly use full data statistics

- Global vectors for word embedding
  - Purpose is to combine statistical benefits of global matrix factorization methods with analogy benefits of context window based methods

- Objective
  - Let X be a word / word (context) count matrix
  - Let w_i and b_i be the word vector and bias of the ith word
  - Let w_j and b_j be the word vector and bias of the jth word
  - Let f() be a weighting function that doesn't overweight rare or frequent word occurrences
  - $J = \Sigma_{i,j} f(X_{ij}) (w_i^T w_j + b_i + b_j - \log(X_{ij}))^2$

Visualization Via t-SNE

- Word embeddings can be vectors of ~ 1000 dimensions
  - So how were they plotted in 2 or 3 dimensions a few slides prior to this?
  - t–SNE is a common method for mapping high dimension vectors to 2 or 3 dimensions for plotting
- Compute probabilities for high dimension vectors that are ∝ to the similarities of pairs of objects
  - Let x_i be a vector in the high dimension input space
  - $p_{j|i} = \exp(-||x_i - x_j||^2/(2\sigma_i^2)) / \Sigma_{k \neq i} \exp(-||x_i - x_k||^2/(2\sigma_i^2))$
  - $p_{ij} = (p_{j|i} + p_{i|j}) / (2N)$
- Compute similarities for low dimension vectors
  - Let y_i be the vector in the low dimension output space
  - $q_{ij} = \exp(1 + ||y_i - y_j||^2)^{-1} / \Sigma_{k \neq l} \exp(1 + ||y_i - y_l||^2)^{-1}$
- Adapt y_i to minimize the KL divergence using gradient descent
  - $KL(P||Q) = \Sigma_{i \neq j}\, p_{ij} \log (p_{ij} / q_{ij})$

Evaluation

- All the embeddings discussed in this section start with a word and end up with a dense vector
  - So which is better?

- Some intrinsic metrics used for comparisons based on the original intent of capturing language meaning
  - Word similarity
    - Data is composed of pairs of words with similarity scores assigned by humans, goal is be to match score with something like cosine between vectors
    - Ex: SimLex999, MEN, WordSimilarity353 and RareWords
  - Word analogy
    - Data is composed of quadruples generated by humans such as king queen man woman, goal is be to determine 1 from the other 3
    - Ex: WordRep
  - Sentence
    - Data is composed of sentences with scores assigned by humans, goal is to match score
    - Ex: Stanford Sentiment Tree-bank and News20
  - Single word
    - Data is composed of single words with classes assigned by humans, goal is to match class
    - Ex tasks: POS tagging, sentiment, color, WordNet synset

- Word embedding benchmarks
  - https://github.com/kudkudak/word-embeddings-benchmarks

Evaluation criteria from https://arxiv.org/abs/1702.02170

Evaluation

- But really the main thing that matters is extrinsic to the embedding
  - Does the subsequent downstream language task perform better or worse with 1 mapping or another
- This highlights that the cost function optimized to find the mapping is not the final cost function that matters

Task Specific Optimization

- The output of the previously discussed word embedding methods is a matrix that maps very large 1 hot vectors to a much smaller length ~ 100 – 1000 dense vectors
- This matrix is typically generated via an exceedingly large offline text and is well optimized for that text
- However, this mapping choice is not necessarily optimal for the subsequent downstream application
  - 1 of the keys to xNN success in many applications that we've looked at is end to end training
- Task specific optimization seeks to refine an existing mapping or optimize a new mapping from scratch by incorporating the mapping as a trainable component in the task specific network structure

Language Modeling

Word Based Language Modeling

- Assign a probability to a sequence of words
  - $P(w_{n-1}, w_{n-2}, \ldots, w_1, w_0) = P(w_{n-1} \mid w_{n-2}, \ldots, w_1, w_0) \ldots P(w_1 \mid w_0)\, P(w_0)$
    - Remember the chain rule of conditional probability from the probability lecture
  - So we can create a language model from learning to predict the next word for all different length sequences (this is important)
    - $P(w_{n-1} \mid w_{n-2}, \ldots, w_1, w_0) = P(w_{n-1}, w_{n-2}, \ldots, w_1, w_0) / P(w_{n-2}, \ldots, w_1, w_0)$

- Next word prediction is a fundamental component of other language tasks
  - We'll see it in text to text translation
  - We'll see it in speech to text transduction

Data

- Penn Treebank
- Billion word corpus
- WikiText

Count Based N Gram Models

- Predict the Nth word in a N word sequence from the previous N – 1 words
- Strategy: estimate probabilities from counting in text and building up recursively
  - $P(w_{n-1} \mid w_{n-2}, \ldots, w_1, w_0) = P(w_{n-1}, w_{n-2}, \ldots, w_1, w_0) / P(w_{n-2}, \ldots, w_1, w_0)$
- Problem: what if a word doesn't occur in the training text
  - Solution: smoothing
  - Assign a small non 0 probability to all words
- Problem: what if the N – 1 sequence never occurs
  - Solution: back off
  - Use N – 2 sequence instead or more generally interpolate all different lengths
  - $P(w_{n-1} \mid w_{n-2}, \ldots, w_1, w_0) \approx c_0\, P(w_{n-1} \mid w_{n-2}, \ldots, w_1, w_0) + c_1\, P(w_{n-1} \mid w_{n-2}, \ldots, w_1) + \ldots + c_{n-1}\, P(w_{n-1})$
  - Where $c_0 + c_1 + \ldots + c_{n-1} = 1$

Neural Language Models

Green arrows represent matrix vector multiplication + bias, dotted are optional, first is table lookup word embedding

- Why use something other than a N gram language model?
  - N gram models are very large, sometimes implementation considerations prevent their use
  - xNNs are good at predicting stuff, perhaps we can find a more compact representation that also allows for larger values of N

- Can create neural language models using all of the network types that we've previously seen
  - NN, RNNs, CNNs, self attention
  - Key is to understand the range of information that they work over and how they combine that information (the exact same considerations as usual)

- Basic idea for NN based method
  - Individually encode each of N – 1 words into vectors via an embedding layer
  - Concatenate the vectors and (potentially) have multiple intermediate layers
  - Classifier head to predict a pdf of the Nth word (this softmax complexity is large)

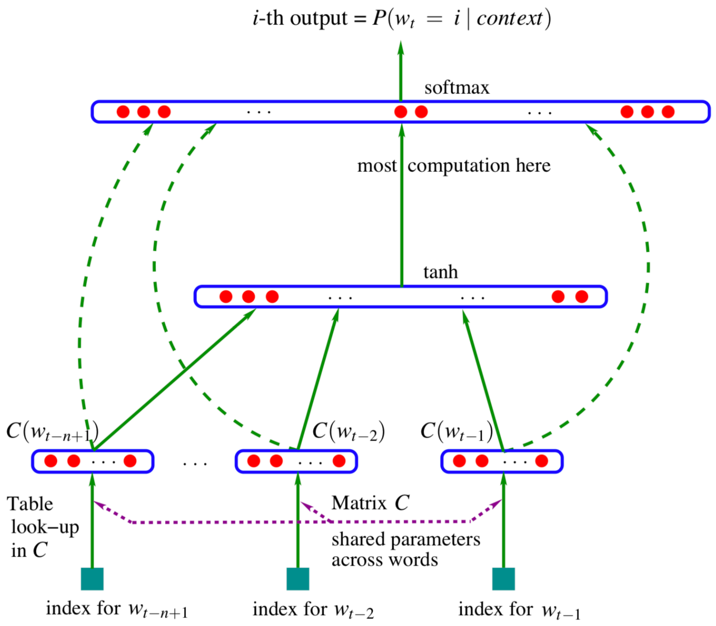


Figure from http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf

NN Language Model Example

Example of standard neural network language model training predicting the next word from the previous 2 words

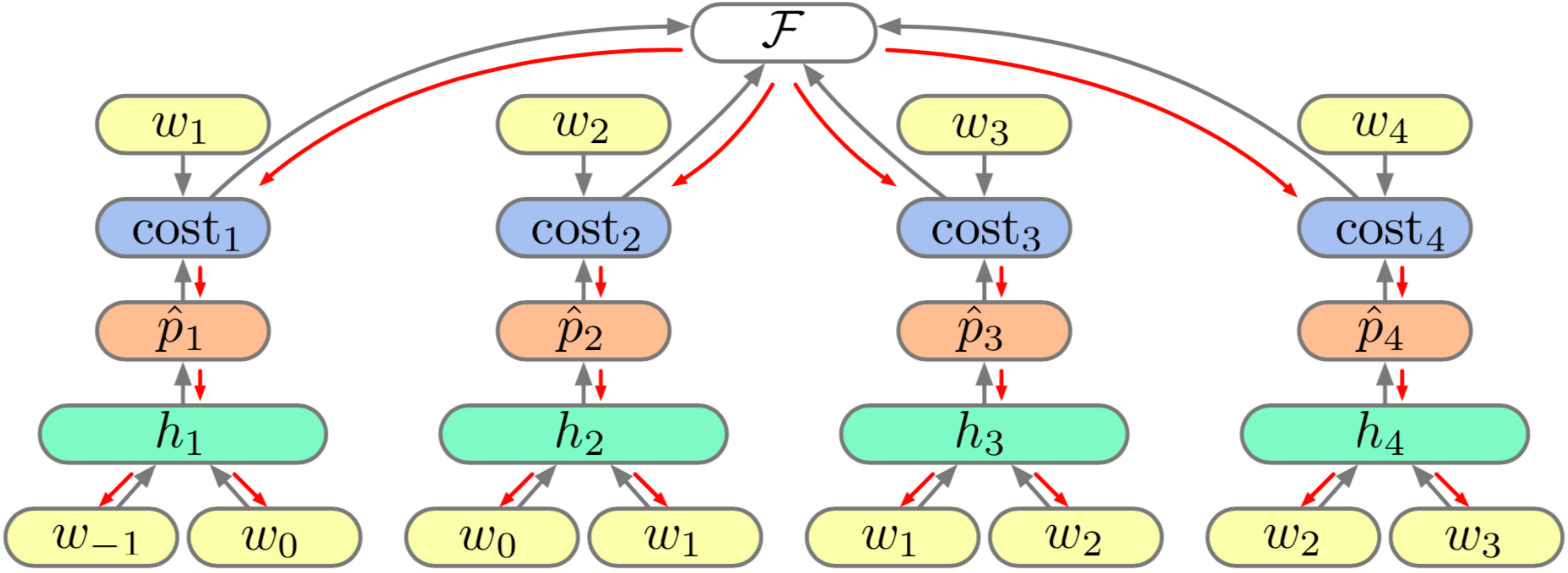


Figure from https://github.com/oxford-cs-deepnlp-2017/lectures/blob/master/Lecture%203%20-%20Language%20Modelling%20and%20RNNs%20Part%201.pdf

RNN Language Model Example

Example of recurrent neural network language model training predicting the next word from the previous history of words

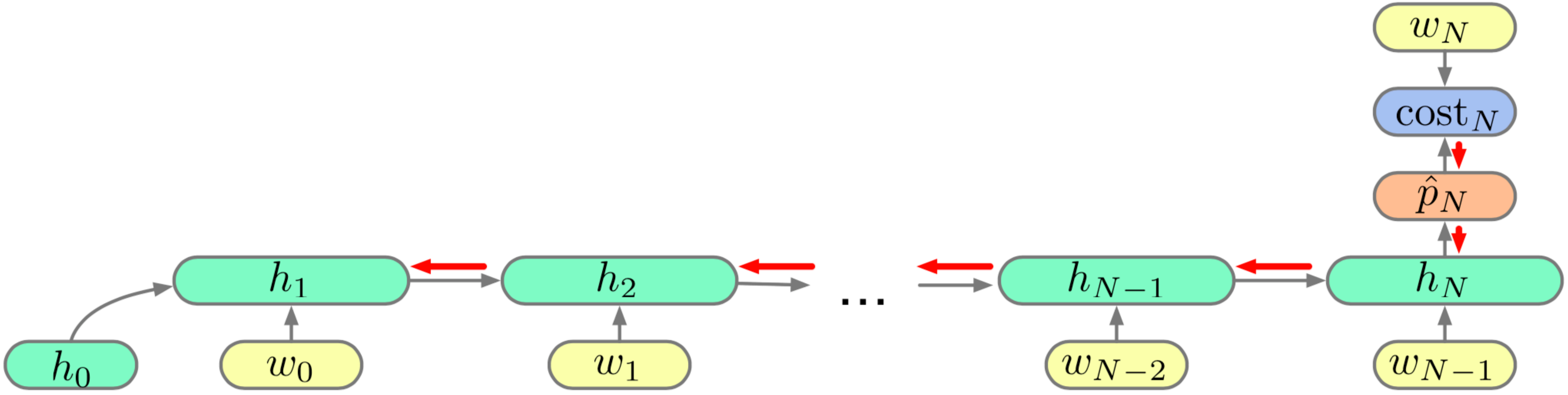


Figure from https://github.com/oxford-cs-deepnlp-2017/lectures/blob/master/Lecture%204%20-%20Language%20Modelling%20and%20RNNs%20Part%202.pdf

CNN Language Model Example

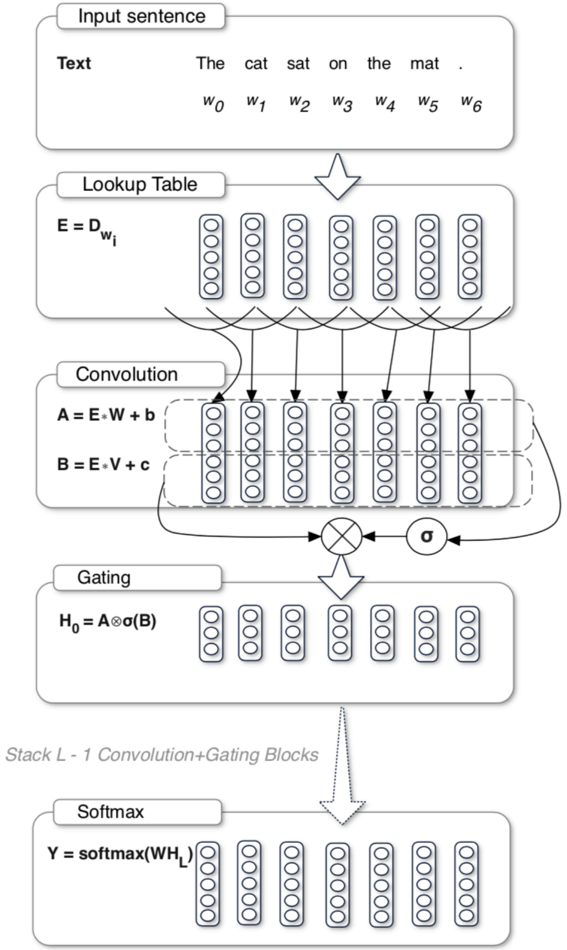


Figure from https://arxiv.org/pdf/1612.08083.pdf

Evaluation

- Want a measure of how well the language model predicts a sample
  - Remember from info theory that cross entropy $H(p, q) = -\Sigma_x p(x) \log_2(q(x))$ fell out of Kullback–Leibler divergence as a method of comparing pmfs
  - Let p(x) be the true distribution of words and q(x) be the distribution predicted by the model
  - Unfortunately, the true distribution p(x) is unknown so cross entropy is estimated as H(training data, q) = $-\Sigma_{xi} (1/N) \log_2(q(x_i))$ where N is the size of the testing data

- Perplexity is defined as $2^{H(\text{training data, q})}$
  - Interpretation of how many different equally probable words can follow as the next word
  - A low perplexity is the goal (implying a low cross entropy, requires a good pmf match)

- However, just like for the case of word embeddings, the better evaluation of a language model is how well it helps with the subsequent downstream task
  - How much does it improve speech to text transduction accuracy?
  - How much does it improve text to text translation accuracy?

Character Based Language Modeling

- A challenge of language modelling is the size of the vocabulary (lots of words)
- Instead of building a word based language model, it's possible to create a character based language model
  - Potentially a much much smaller model (number of characters is << number of words)
  - Able to capture sub word relationships
  - But perhaps not as good at word relationships as a word optimized model
  - Note: this is used in the RNN transformer for speech to text
- A common word piece + all characters is another option
- Just something to consider if you're thinking about highly resource constrained applications

A Little Bit Of Fun

- You can train a language model using specific text to learn to predict words in the style of that text
- You can then use a language model to generate new text via feeding in it's previous outputs
  - For a N gram language model, use the Nth predicted word output as the most recent input to predict the N + 1 th word (similar strategy for a RNN language model)
- Things tend to go off the rails after a bit but it's interesting to think about and leads to the view on the next slide
  - Until very recently, the train was quickly off the track
  - Now ... there are much longer much more coherent sequence of text that can be generated

Sentence Embeddings

Language Specific Sentence Embedding

- This is an active area of research, pre training on a language modeling task to create embeddings (features) that can be used with simple decoders via fine tuning for many additional language tasks
  - Sometimes referred to as language's ImageNet + transfer learning (of the trained encoder) moment
  - Looking at a full sentence gives better context / understanding than 1 word at a time

- Basic strategy
  - Input sentence with words mapped to vectors via a common word to vector model
  - Encoder, typically Transformer based (larger == better)
  - Final output features viewed as an input embedding
  - Decoder for an unsupervised task (predict the next word, predict a masked word, predict if a sentence is the next sentence)
  - Unsupervised training on a dataset (larger == better)
  - Remove the unsupervised task decoder
  - Add a new decoder for new task and fine tune on the new task

Different Mixings Of Data And Training Tasks

- A few examples
  - Semi-supervised sequence learning
  - Generative pre-training I and II (GPT)
  - Embeddings from language models (ELMo)
  - Universal language model fine tuning (ULMFit)
  - Bi directional encoder representation from transformers (BERT)

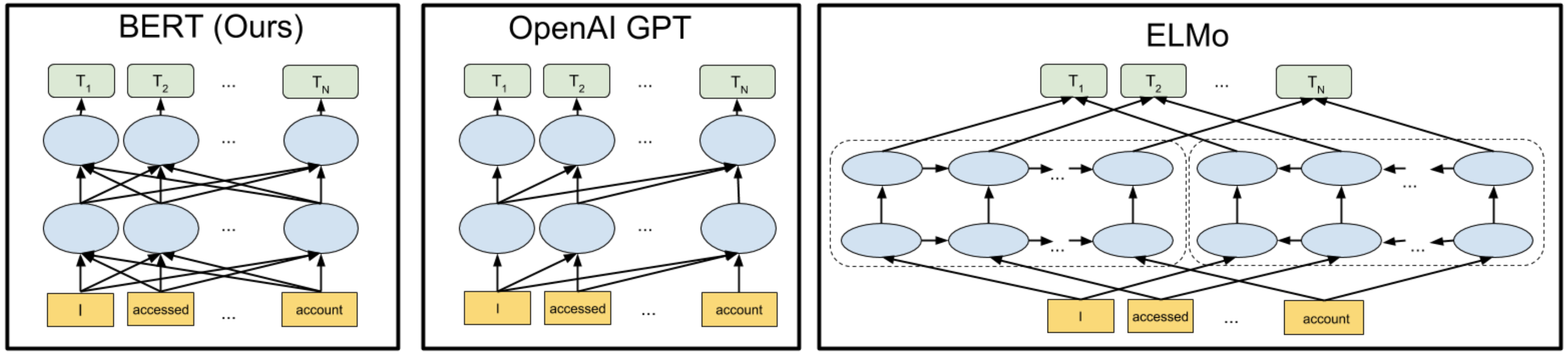


Figure from https://arxiv.org/abs/1810.04805

Example: BERT

The input embedding consists of word piece embeddings, segment (A / B) allowing 2 sentences and absolute position; masked word prediction and is a sentence the next sentence tasks are used unsupervised training

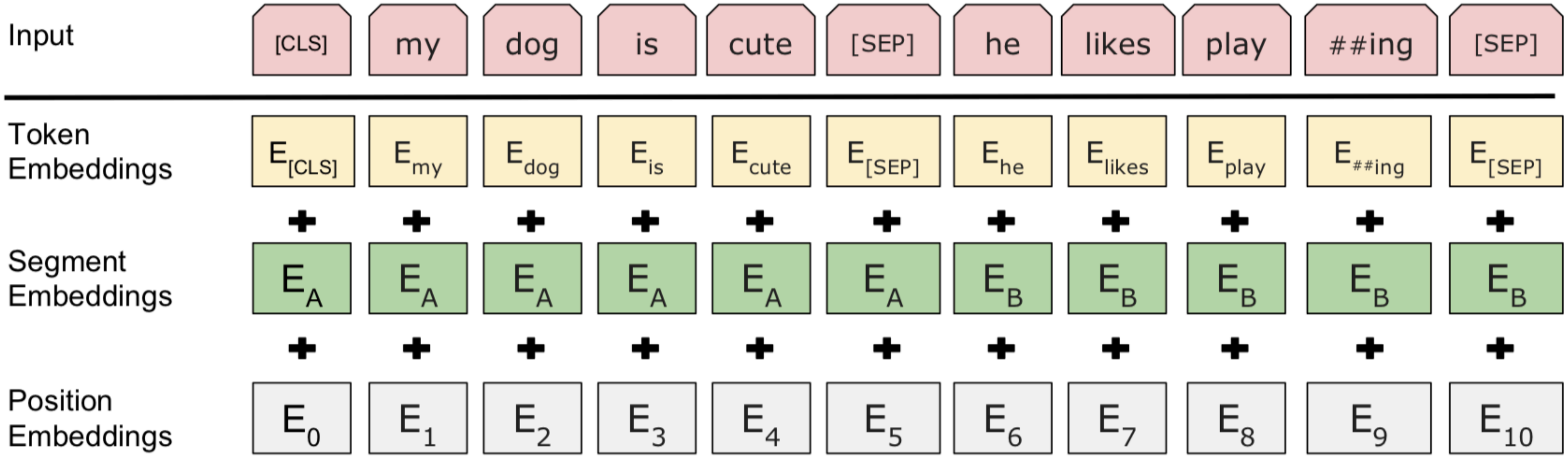
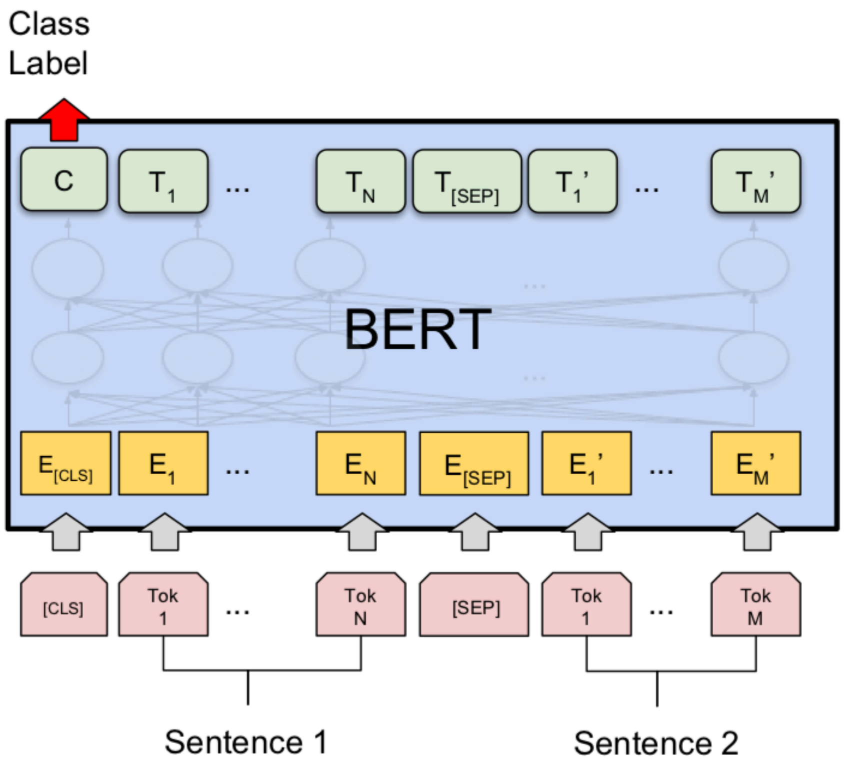


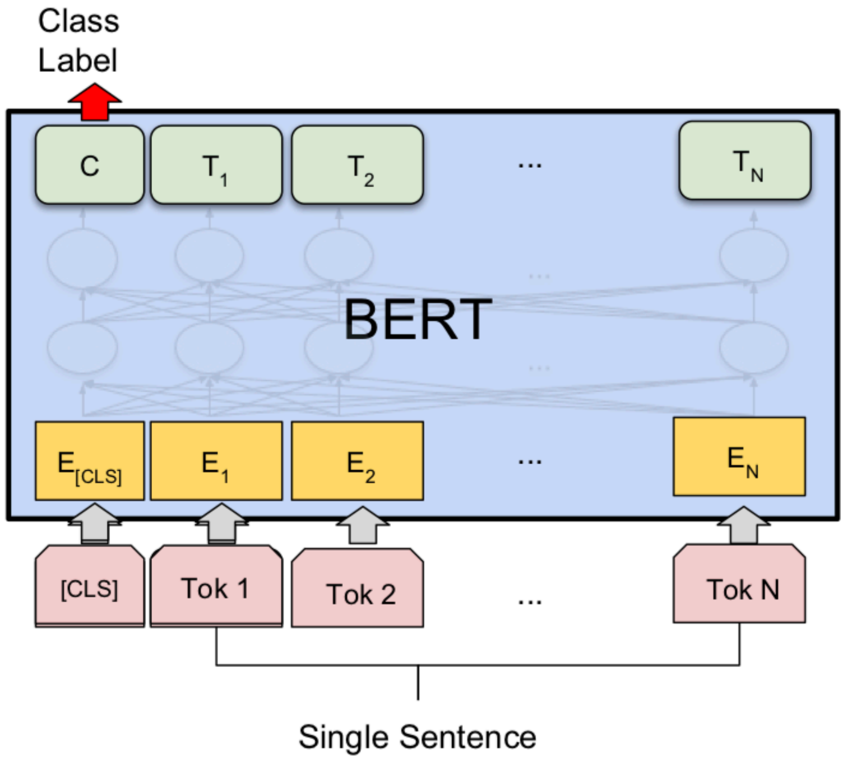
Figure from https://arxiv.org/abs/1810.04805

Example: BERT

The resulting embedded inputs are used for multiple language tasks



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

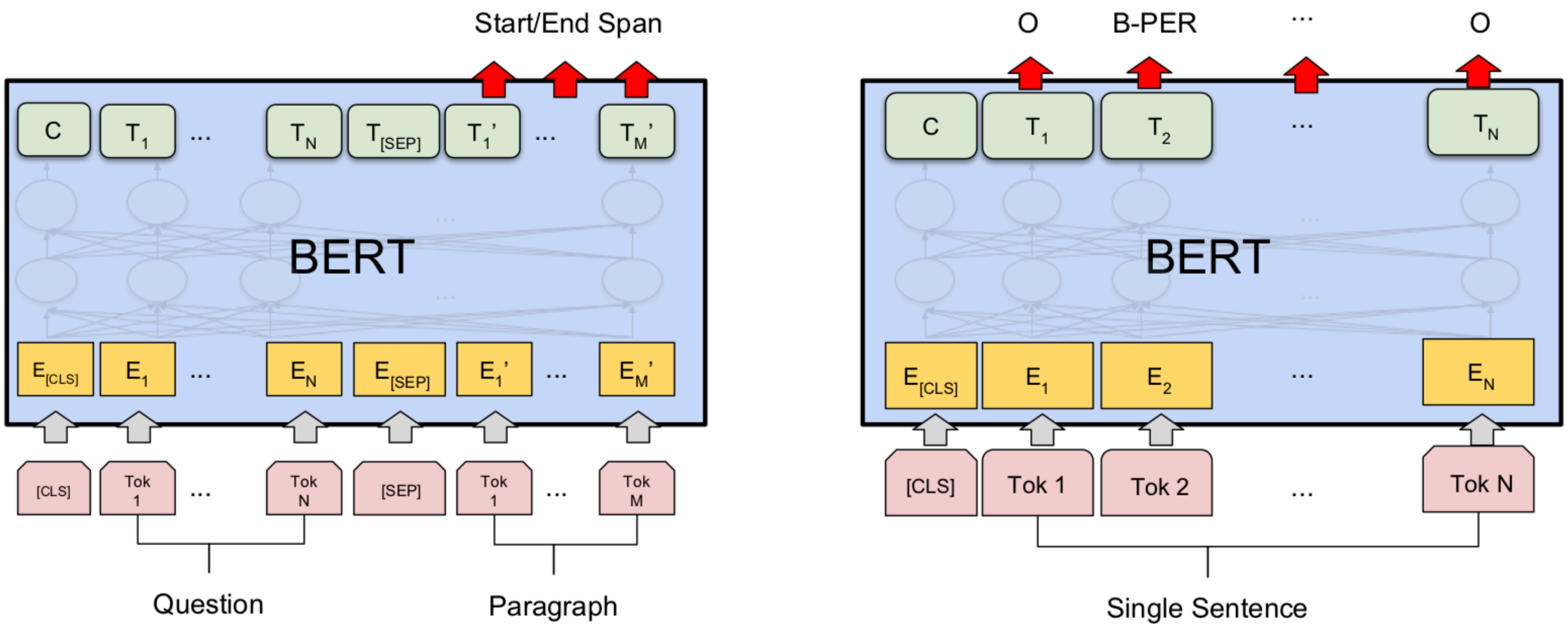


(b) Single Sentence Classification Tasks: SST-2, CoLA

Figure from https://arxiv.org/abs/1810.04805

Example: BERT

The resulting embedded inputs are used for multiple language tasks



(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Figure from https://arxiv.org/abs/1810.04805

Language Agnostic Sentence Embedding

- If the resulting sentence embedding represents the meaning, can the embedding be made language agnostic?
  - Think: mapping different (language) representations of the same thought all to the same embedding
  - The benefit of this is that it would simplify transfer learning of language tasks from 1 language to another; especially helpful for low resource languages

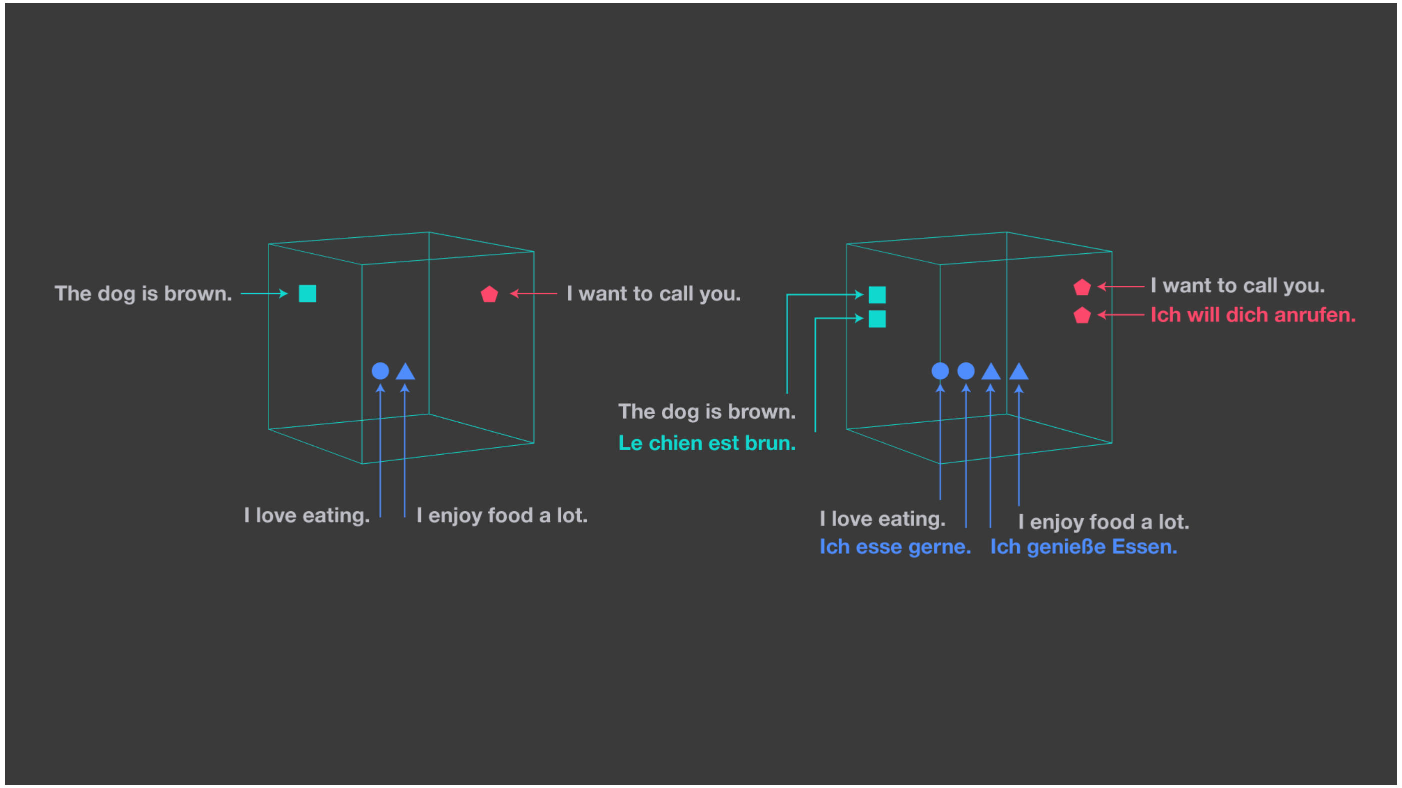
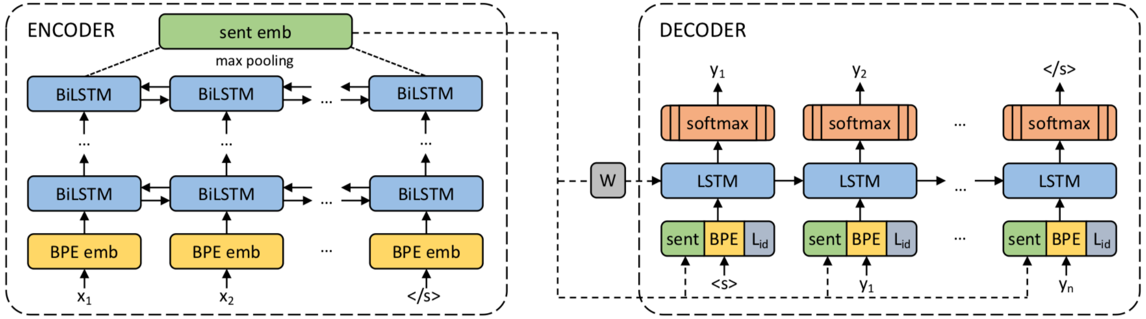


Figure from https://code.fb.com/ai-research/laser-multilingual-sentence-embeddings/

Example: LASER

Language agnostic sentence representations (not light amplification by stimulated emission of radiation)

- Goal
  - Learn input embeddings that are agnostic to language and universal in applicability to task
- Training
  - Encoder → vector → decoder architecture
  - Use parallel text from 2 high resource languages, English and Spanish
  - Alternate between **all combinations** of input language and output language
- Task transfer (currently supports 93 languages)
  - Remove decoder
  - ~ train language specific task specific decoder



References

- Zero-shot transfer across 93 languages: open-sourcing enhanced LASER library
  - https://code.fb.com/ai-research/laser-multilingual-sentence-embeddings/
- LASER Language-Agnostic SEntence Representations
  - https://github.com/facebookresearch/LASER
- Learning joint multilingual sentence representations with neural machine translation
  - https://arxiv.org/abs/1704.04154
- A corpus for multilingual document classification in eight languages
  - https://arxiv.org/abs/1805.09821
- Filtering and mining parallel data in a joint multilingual space
  - https://arxiv.org/abs/1805.09822
- XNLI: evaluating cross-lingual sentence representations
  - https://arxiv.org/abs/1809.05053
- Margin-based parallel corpus mining with multilingual sentence embeddings
  - https://arxiv.org/abs/1811.01136
- Massively multilingual sentence embeddings for zero-shot cross-lingual transfer and beyond
  - https://arxiv.org/abs/1812.10464

Figure from https://arxiv.org/abs/1812.10464

Going Forward

- Themes / directions of active research
  - Modifying the architecture to improve the accuracy on the downstream task
  - Modifying the architecture to improve performance via reducing memory, reducing compute, ...
  - Modifying the training to improve the accuracy of a given architecture
  - Modifying the training to improve the training time / convergence

- Tracking performance
  - GLUE / SuperGLUE benchmark and leaderboard
  - https://gluebenchmark.com
  - https://gluebenchmark.com/leaderboard/
  - https://super.gluebenchmark.com/leaderboard

Some Recent References

- Multi-task deep neural networks for natural language understanding
  - https://www.aclweb.org/anthology/P19-1441/
  - https://github.com/namisan/mt-dnn
- Character-level language modeling with deeper self-attention
  - https://arxiv.org/abs/1808.04444
- Transformer-XL: attentive language models beyond a fixed-length context
  - https://arxiv.org/abs/1901.02860
- Improving multi-task deep neural networks via knowledge distillation for natural language understanding
  - https://arxiv.org/abs/1904.09482
- How to fine-tune BERT for text classification?
  - https://arxiv.org/abs/1905.05583
- XLNet: generalized autoregressive pretraining for language understanding
  - https://arxiv.org/abs/1906.08237
- Large memory layers with product keys
  - https://arxiv.org/abs/1907.05242

Some Recent References

- RoBERTa: a robustly optimized BERT pretraining approach
  - https://arxiv.org/abs/1907.11692
- ERNIE 2.0: a continual pre-training framework for language understanding
  - https://arxiv.org/abs/1907.12412
  - https://github.com/PaddlePaddle/ERNIE
- StructBERT: incorporating language structures into pre-training for deep language understanding
  - https://arxiv.org/abs/1908.04577
- Sentence-BERT: sentence embeddings using siamese BERT-networks
  - https://arxiv.org/abs/1908.10084
- ALBERT: a lite BERT for self-supervised learning of language representations
  - https://arxiv.org/abs/1909.11942
  - https://github.com/google-research/ALBERT
- Exploring the limits of transfer learning with a unified text-to-text transformer
  - https://arxiv.org/abs/1910.10683
  - https://github.com/google-research/text-to-text-transfer-transformer
- SMART: robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization
  - https://arxiv.org/abs/1911.03437

A Little More Fun

- Pun generation with surprise
  - https://arxiv.org/abs/1904.06828
- There's a subreddit populated entirely by AI personifications of other subreddits
  - https://www.theverge.com/2019/6/6/18655212/reddit-ai-bots-gpt2-openai-text-artificial-intelligence-subreddit
  - Reddit r/SubSimulatorGPT2
    - https://www.reddit.com/r/SubSimulatorGPT2/
    - https://www.reddit.com/r/SubSimulatorGPT2/comments/btfhks/what_is_rsubsimulatorgpt2/

Conditional Modeling

Model → Conditional Model

A single framework for thinking about speech to text, text to speech, language translation and other problems with an input and feedback in the prediction

- Model
  - Assigns probabilities to a sequence of elements y_i (words, audio samples, ...)
    - $P(y_{n-1}, y_{n-2}, \ldots, y_1, y_0) = P(y_{n-1} \mid y_{n-2}, \ldots, y_1, y_0) \ldots P(y_1 \mid y_0)\, P(y_0)$
  - The key to creating a model is next element prediction (next word, next audio sample, ...) given previous elements
    - $P(y_{n-1} \mid y_{n-2}, \ldots, y_1, y_0) = P(y_{n-1}, y_{n-2}, \ldots, y_1, y_0) / P(y_{n-2}, \ldots, y_1, y_0)$

- It's possible to cast a large number of problems as learning a model using output data (next element prediction) then focusing / biasing the prediction by conditioning the model on input data **x**
  - $P(y_{n-1} \mid y_{n-2}, \ldots, y_1, y_0, \mathbf{x})$
  - Effectively, conditioning on the input data makes the next element prediction less uniform / more spikey (ideally 1 hot like)
  - Reduces the entropy of the conditional pmf
  - Needs input output data pairs for training

- During testing previous true outputs $y_{n-2}, \ldots, y_1, y_0$ are replaced with previous predicted outputs
  - $P(y_{n-1} \mid y_{n-2}^{hat}, \ldots, y_1^{hat}, y_0^{hat}, \mathbf{x})$
  - Use beam search or some similar variant to reduce error feedback
  - Even better if **x** contributes strongly to the prediction as that helps prevent error feedback effects

Conditional Model For Speech To Text

This is covered in the speech lecture

- Learn a model for text that can predict the next phoneme, grapheme / character, word piece or word given previous phonemes, graphemes / characters, word pieces or words
  - This model can be optimized for specific or general types of text by training on that type of text
- Then focus / bias the text predictions via conditioning on features generated from a specific speech signal
  - This creates a conditional model that produces text corresponding to the given speech signal
- In equations (the network approximates this pmf)
  - Notation: y_i is the text; $\mathbf{x}$ is the speech signal
  - Text model: $P(y_{n-1} \mid y_{n-2}, \ldots, y_1, y_0)$
  - Conditioned on features from speech: $P(y_{n-1} \mid y_{n-2}, \ldots, y_1, y_0, \mathbf{x})$
  - Using previous predicted outputs: $P(y_{n-1} \mid y_{n-2}^{hat}, \ldots, y_1^{hat}, y_0^{hat}, \mathbf{x})$

Conditional Model For Text To Audio

This is covered in the speech lecture

- Learn a model for audio that can predict the next audio sample given previous audio samples
    - This model can be optimized for specific or general types of audio by training on that type of audio
    - Ex: human speech, 1980s hairspray metal, ...

- Then focus / bias the audio sample predictions via conditioning on features generated from a specific text
    - This creates a conditional model that produces audio (speech, music) corresponding to the given text (words, instruments)
    - Note that it's possible to condition on more than 1 thing (e.g., words + voice characteristics from a specific speaker to create a conditional model that produces speech corresponding to the specific words in the voice of the specific speaker)

- In equations (the network approximates this pmf)
    - Notation: y_i is the audio samples; $\mathbf{x}$ is the text signal
    - Audio model: $P(y_{n-1} \mid y_{n-2}, \ldots, y_1, y_0)$
    - Conditioned on features from text: $P(y_{n-1} \mid y_{n-2}, \ldots, y_1, y_0, \mathbf{x})$
    - Using previous predicted outputs: $P(y_{n-1} \mid y_{n-2}^{hat}, \ldots, y_1^{hat}, y_0^{hat}, \mathbf{x})$

Conditional Model For Language Translation

This is covered in the next section

- Learn a model for language 2 that can predict the next word in language 2 given previous words in language 2
  - This model can be optimized for specific or general text from language 2 by training on that type of text from language 2

- Then focus / bias the language 2 predictions via conditioning on features generated from specific text of language 1
  - This creates a conditional model that produces text of language 2 corresponding to the given text of language 1

- In equations (the network approximates this pmf)
  - Notation: y_i are the words in language 2; $\mathbf{x}$ are the text features in language 1
  - Language 2 model: $P(y_{n-1} \mid y_{n-2}, \ldots, y_1, y_0)$
  - Conditioned on features from language 1: $P(y_{n-1} \mid y_{n-2}, \ldots, y_1, y_0, \mathbf{x})$
  - Using previous predicted outputs: $P(y_{n-1} \mid y_{n-2}^{hat}, \ldots, y_1^{hat}, y_0^{hat}, \mathbf{x})$

Translation

Goal

- Translate a sentence from 1 language to another
- Specifically: find best sentence y given sentence x (x in language 0, y in language 1)
  - $y^* = \arg\max_y P(y \mid x)$
  - The previous section considered language models
  - Translation can be viewed as a conditional language model
    - Predicts the next word given the previous words
    - Conditioned on the input sentence
- An incomplete history
  - The Rosetta stone
  - Bilingual dictionary
  - …

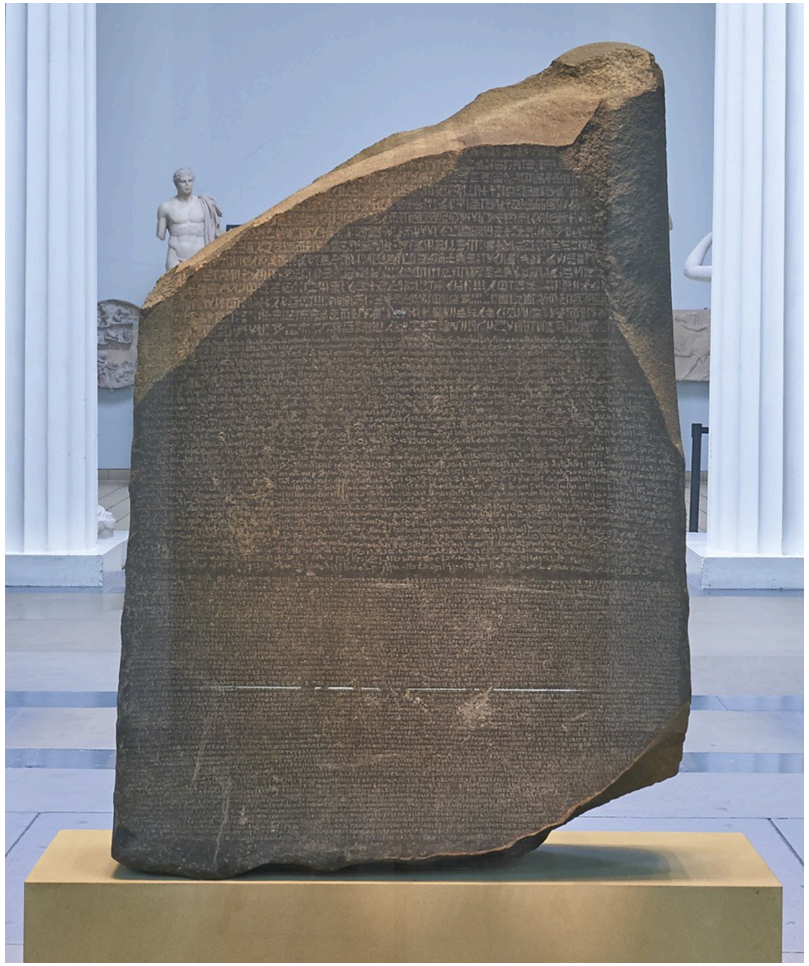


Figure from https://blog.britishmuseum.org/everything-you-ever-wanted-to-know-about-the-rosetta-stone/

Complications

- A laundry list
  - Input output sentence alignment
  - 1 to many mapping of an input sentence to a valid translation
  - How to evaluate the translation
  - Out of vocabulary words
  - Train set vs test set domain mismatch
  - Maintaining context over a long piece of text
  - Low resource language pairs

Encoder Decoder Architecture

The architecture type considered here for translation (note that we've seen this general strategy in vision and speech too)

- How does an encoder decoder architecture translate sentences?
  - You design a network to accomplish a goal
  - So it's useful to think about what the goal of the encoder and decoder is
- The encoder transforms dense word vectors into a vector (seq2seq) or matrix (others) that represents the meaning of the original sentence including a positional word dependence
  - You need to look across the words of a sentence to derive the meaning of the sentence as a human
  - Networks are no different, so the encoder needs to looks across words
- In the vector architecture the meaning for the whole sentence is encoded as a single vector
  - That's a lot to ask of a vector (remember we encode words into vectors of similar length)
  - Implies that the sentence vector length should be >> than the individual word vector lengths but I don't think they are in practice
- In the attention architectures the meaning for the whole sentence is encoded as a matrix
  - This gives more flexibility for encoding meaning with positional word dependence
  - The goal is to encode the meaning from the source language in a way that it can be easily decoded into the target language

Encoder Decoder Architecture

The architecture type considered here for translation (note that we've seen this general strategy in vision and speech too)

- The decoder transforms the encoded meaning with positional word dependence of the source sentence into words of the target language (a pmf over all possible words)

- Note that in practice the encoder and decoder can share some of the tasks in the middle that blur the lines between encoding / decoding meaning and words
    - This is a function of network structure and training

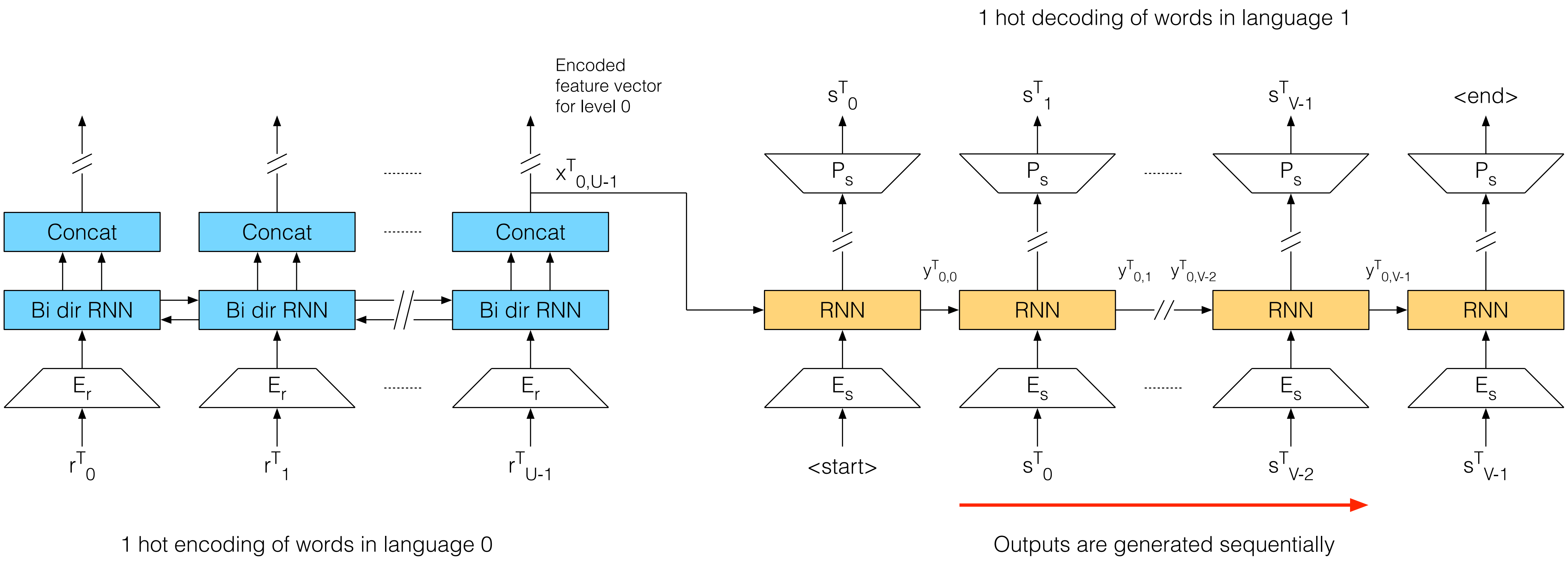
- Uses of attention
    - Self attention at the encoder decides how to look across words or transformations of words to encode words into meaning with a positional dependence
    - Attention between the encoder and decoder decides how to look across the encoded words with positional dependence to generate specific decoder outputs
    - Self attention at the decoder decides how look across encoded meaning with positional word dependence or transformations thereof to generate target words

3 Encoder Decoder Model Types Included Here

- RNN encoder – vector – RNN decoder
  - Source words mapped to vectors
  - Encoder uses a RNN
  - Decoder uses a RNN initialized by the final hidden state of the encoder to recursively generate target word vectors
  - End of sentence output token allows learning variable length translations
- RNN encoder – attention – RNN decoder
  - Decoder uses attention with the encoder hidden states
  - Combines with the hidden state of the decoder RNN and previous output word vector to predict the next output word vector
- Self attention encoder – attention – self attention decoder
  - Organized as a stack of encoders / decoders
  - Uses self attention to compute the encoder and decoder embeddings
  - Includes a positional encoding to track the order of the input and output sequences (since no recurrence)

RNN Encoder – Vector – RNN Decoder

During training known words are used as the decoder input, during testing the previous decoder output word is used as the next decoder input word; reversing the input sentence order tends to improve performance, different RNN style structures can be used for both the encoder and decoder; a highlighted limitation is that the meaning of the entire source sentence is encoded into a single vector, potentially an accuracy bottleneck



Improving Translation Using A Language Model

- Thought train
  - A xNN used for language translation predicts text in language 1 from text in language 0
  - A xNN used for language modeling predicts the next word in language 1 from previous words in language 1
  - A xNN used for language modeling is typically trained on a much larger text than a xNN used for language translation
  - Can we use a language model to improve language translation?
  - Say a bilingual language 0 native speaker translates a sentence from language 0 to language 1. Can a language 1 native speaker improve that translation?

- Basic strategy
  - Instead of using just the 1 most likely word to generate the next word in the language translation decoder, use the most likely N words to generate the most likely next word N times
  - The N most likely words are a combination of the pmf from the language translation xNN and language model xNN

Beam Search Decoding

Beam search works well; recommendation: don’t use overly large beams

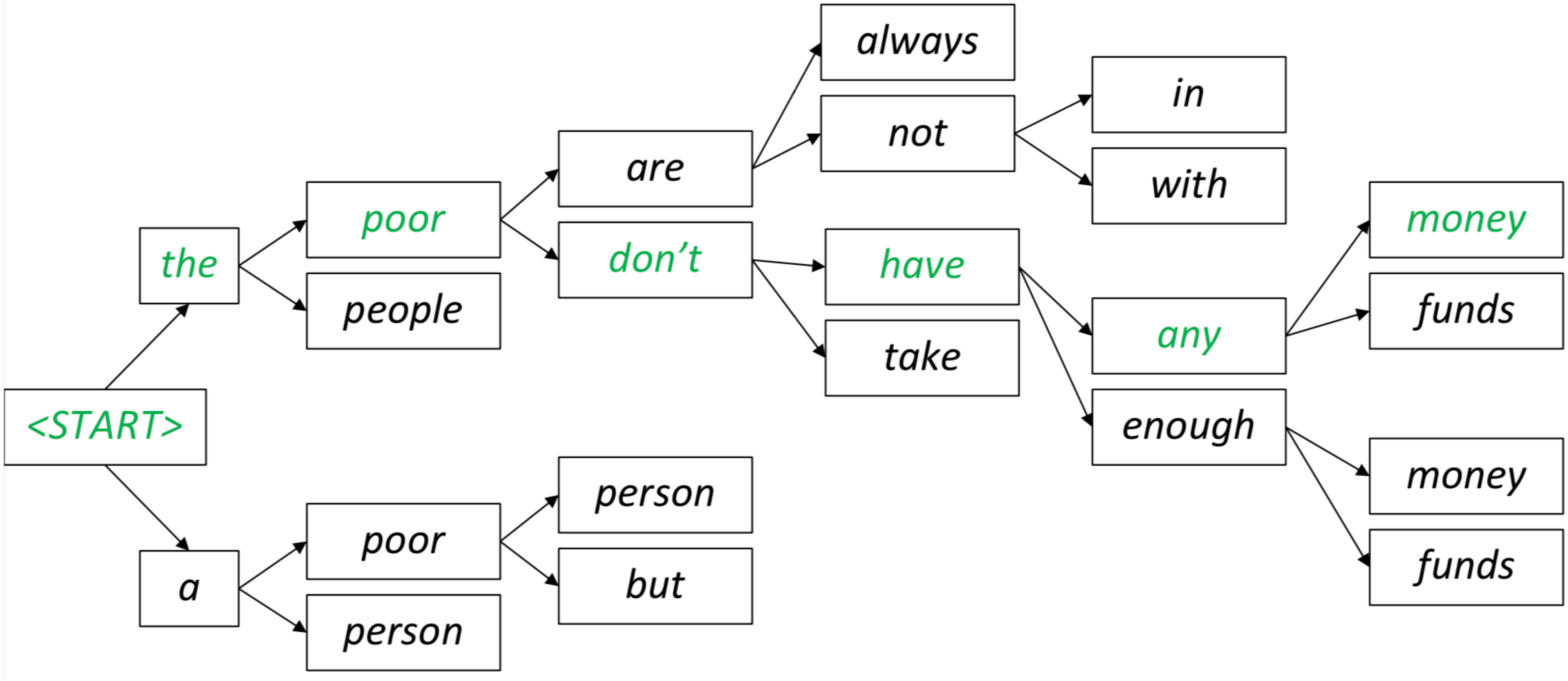


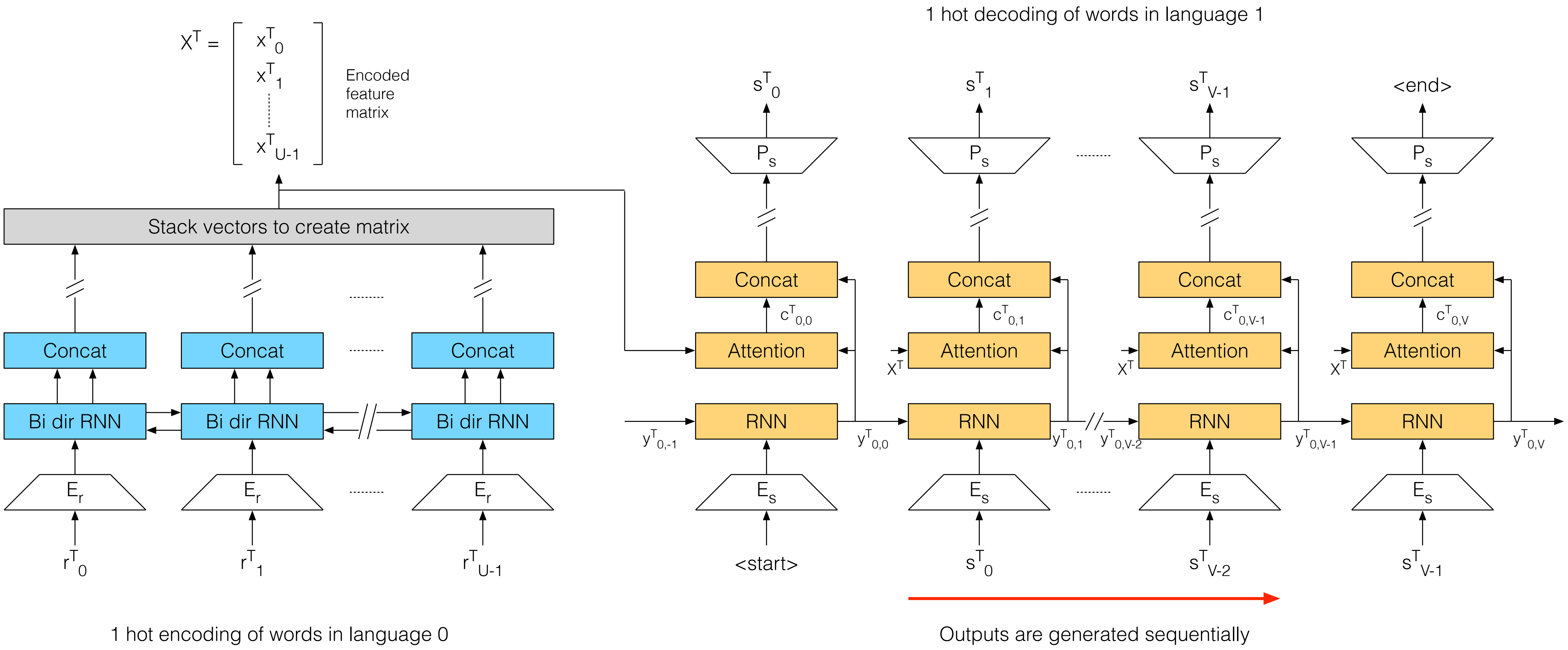
Figure from http://web.stanford.edu/class/cs224n/lectures/lecture10.pdf

Structured Prediction

- Note that test time operation is different than train time operation
  - During train time operation the ground truth is fed back to the decoder input
  - During test time operation the prediction is fed back to the decoder input
  - Want to optimize the sequence performance but the error and operation is optimized for token performance
  - Question: is it possible to optimize train time for full sequence vs token at a time?
- This paper looks at doing that: Classical structured prediction losses for sequence to sequence learning
  - https://arxiv.org/abs/1711.04956
- Some suggestions from the paper
  - Do token level optimization first
  - Maybe use label smoothing
  - Note that this becomes less important as baseline model improves

RNN Encoder – Attention – RNN Decoder

Outputs are generated sequentially via a decoder with access to the full set of encoded features via attention where l = layer, i = output position and context vector $c^T_{l,i}$ = SoftMax(($y^T_{l,i}$ $W_{l,q}$ X) / scale) X^T, with other attention based mapping possible; similar comments as before with respect to network structure apply



Google’s Neural Machine Translation System

Encoder – attention – decoder with a network architecture partitioned across multiple GPUs for performance

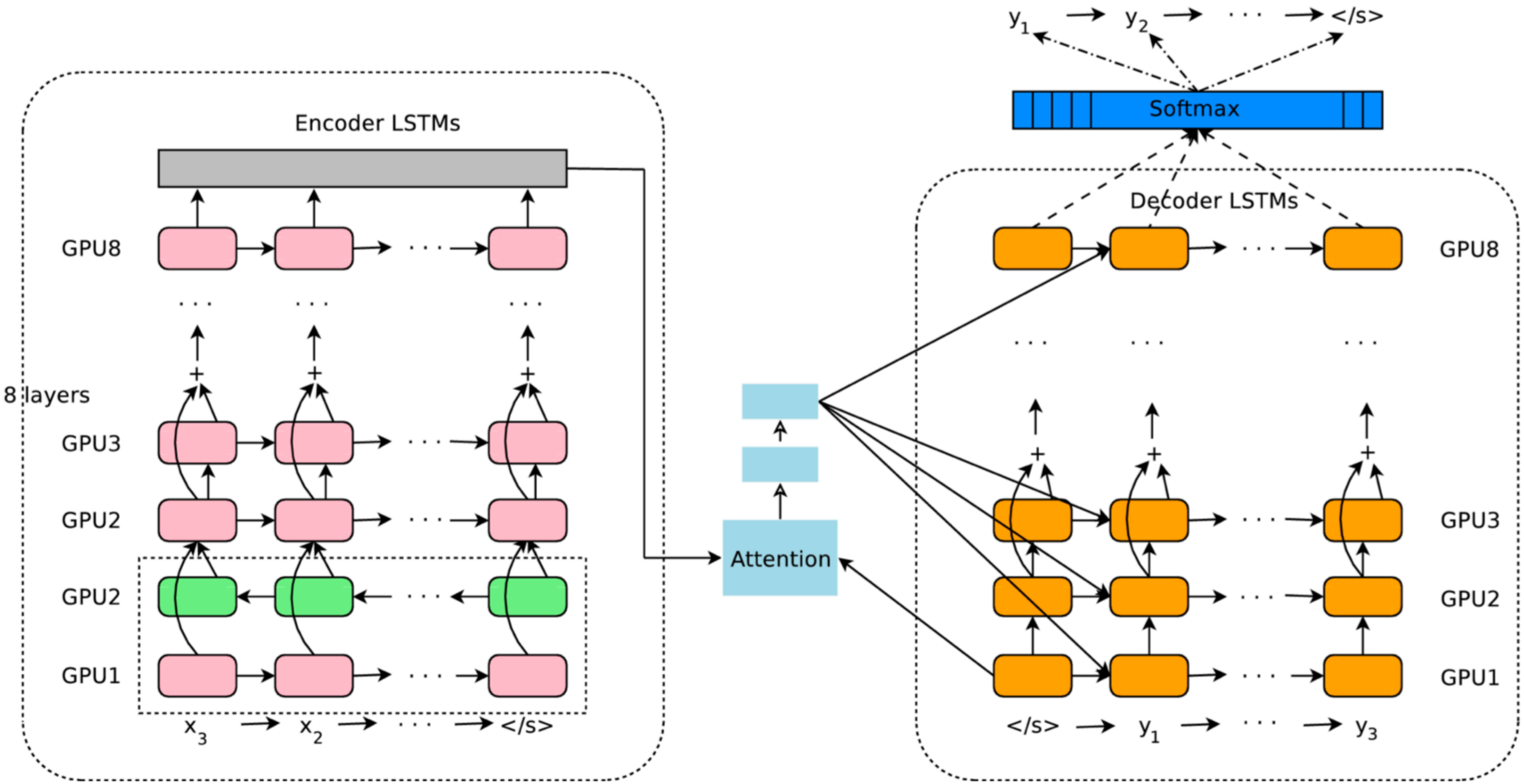
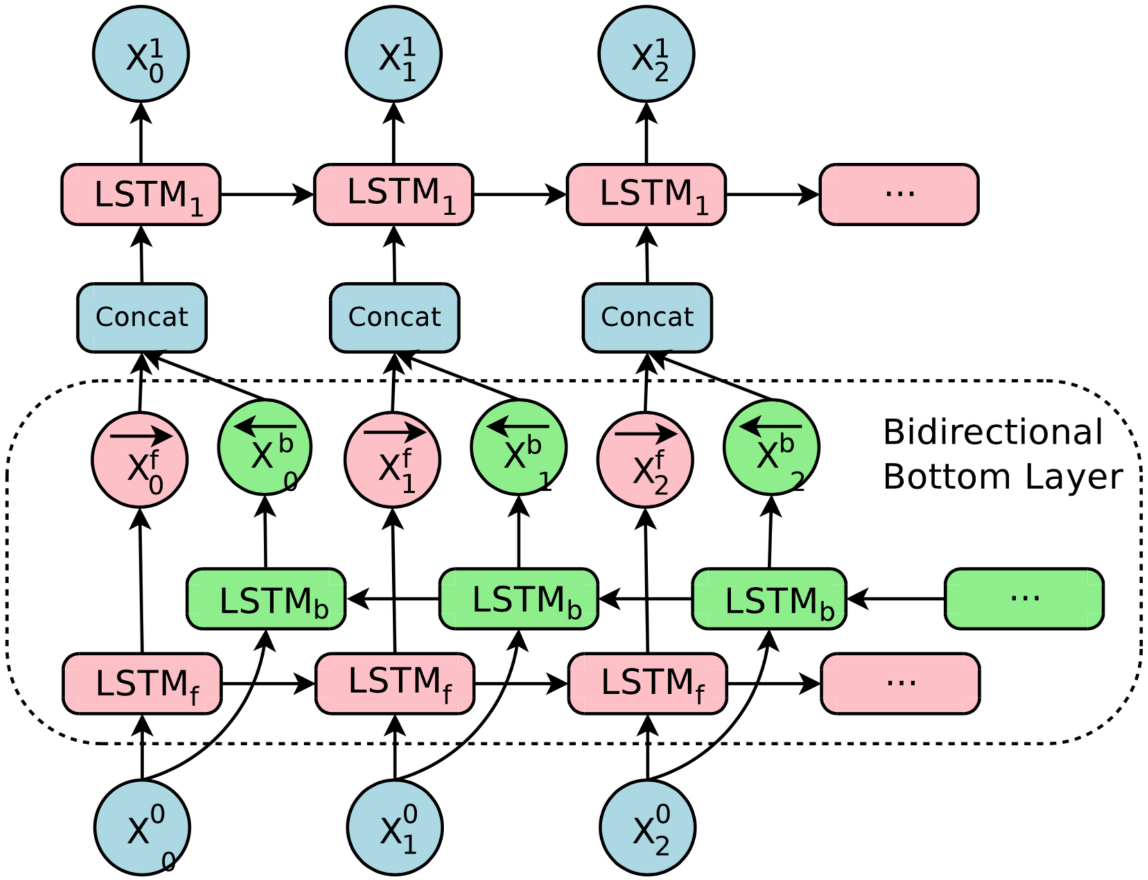


Figure from https://arxiv.org/abs/1609.08144

Google's Neural Machine Translation System

Bottom layer is a bi directional LSTM, subsequent layers are LSTM cells with residual connections



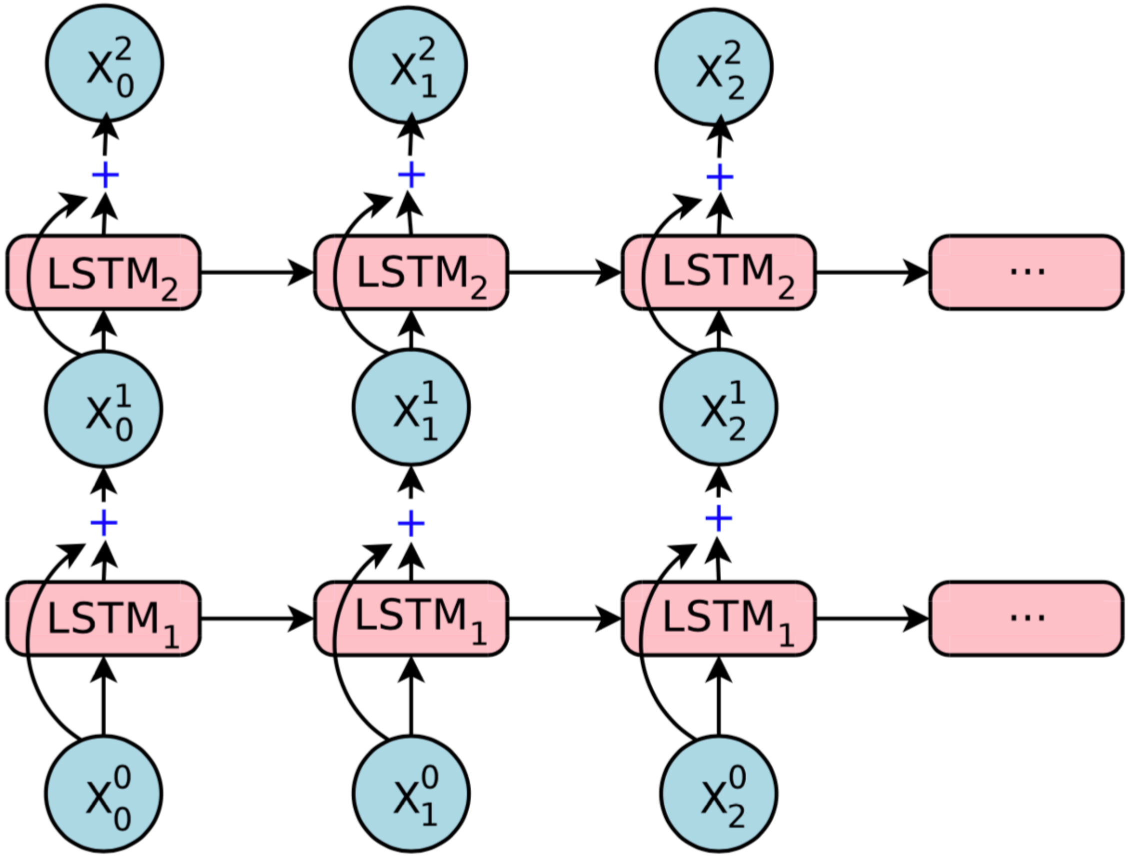


Figure from https://arxiv.org/abs/1609.08144

Architecture Exploration

- Source embedding
  - Performance was relatively agnostic with embedding lengths from 128 – 2048
- RNN cell architecture
  - LSTM was a little better than GRU
  - Standard RNN performed worse
- Encoder and decoder depth
  - Depths from 2 to 8 were explored with and without residual connections
  - Training was difficult with deeper models and needs to be re thought
  - Given current training capabilities, depths of 2 – 4 for the encoder and decoder worked best

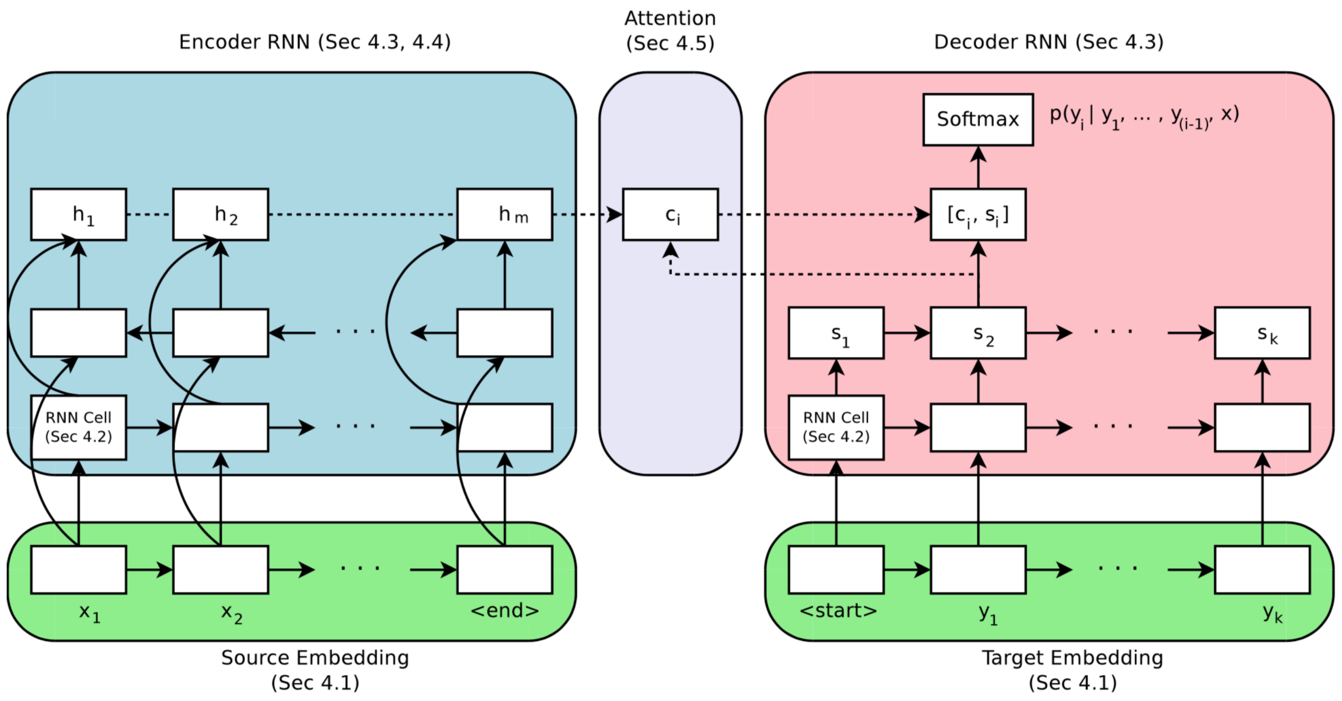


Figure from https://arxiv.org/abs/1703.03906

Architecture Exploration

- Unidirectional vs bi directional encoders
  - Bi directional performed slightly better
- Attention
  - Additive performed slightly better than multiplicative
  - The dimension size from 128 – 1024 didn't matter much
  - Training data indicated that attention played a larger role in the flow of the gradient than allowing the decoder access to the encoded states (as common belief suggests)
  - More thought is needed here
- Beam search
  - A beam width of ~ 10 with a length penalty performed best

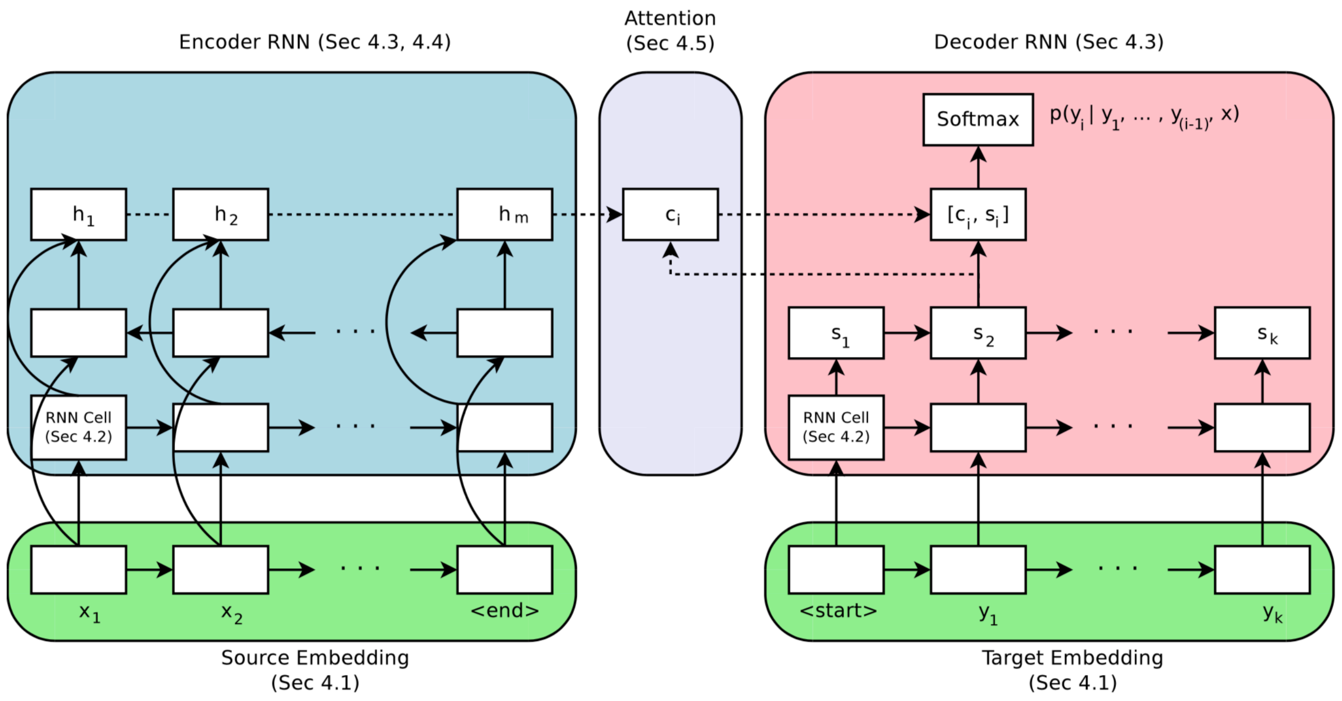


Figure from https://arxiv.org/abs/1703.03906

RNMT+

- More architecture exploration
- Borrows positive parts of multiple architectures
  - Bi directional encoder, uni directional decoder
  - Bi directional layers use concatenation
  - Each layer follows a normalize – transform – dropout – add flow
  - Training uses label smoothing, L2 weight decay, gradient clipping and the Adam optimizer

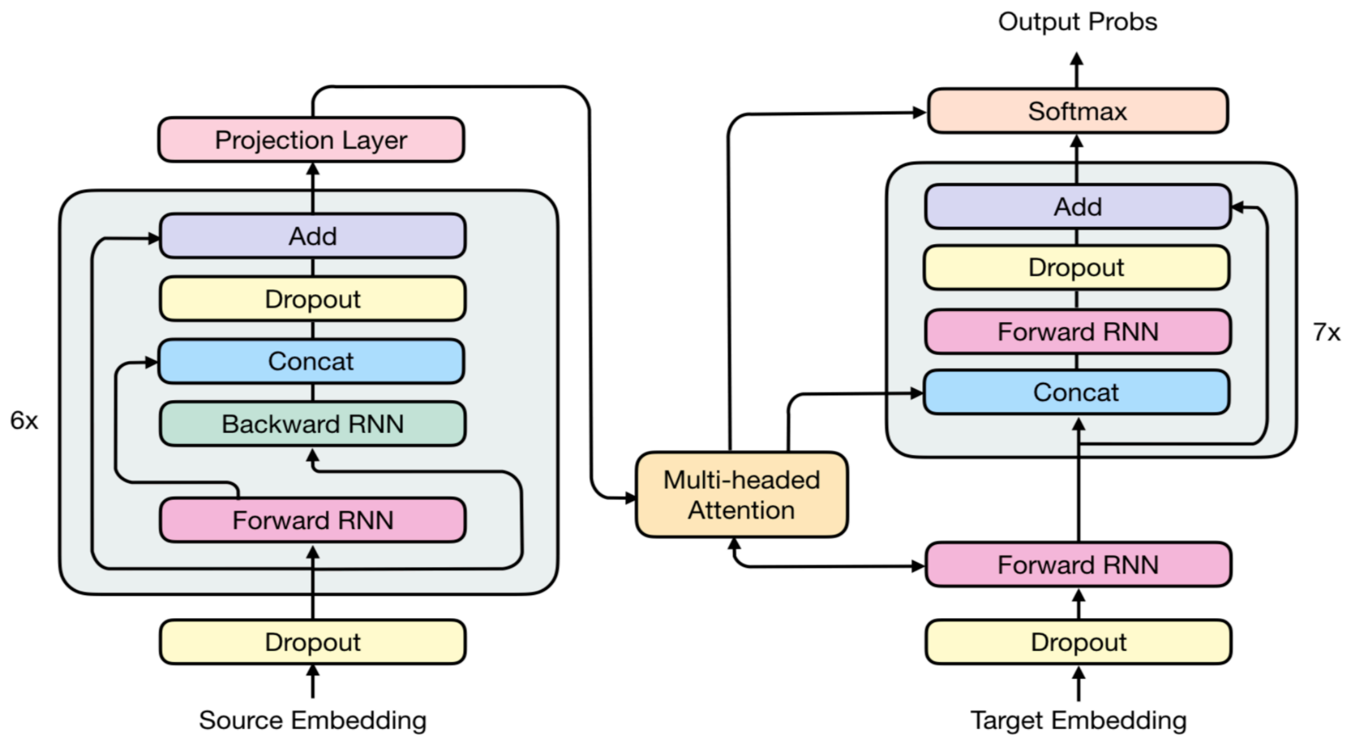


Figure from https://arxiv.org/abs/1703.03906

Self Attention Encoder – Attention – Self Attention Decoder

- Positional encoding
  - No sequential recurrent connections to enforce sequential structure in the output so a positional encoding is added
  - Sin() and cos() vectors with different frequencies are added to the inputs
- Encoder
  - 6 identical layers with 2 parts each
  - Part 1 is a multi head self attention mechanism
  - Part 2 is a fully connected layer
  - Each part includes normalization as y = LayerNorm(x + Part1or2(x)) and dropout
- Decoder
  - 6 identical layers with 3 parts each
  - Part 1 is a multi head self attention mechanism identical to encoder part 1
  - Part 2 is a multi head attention mechanism applied to the encoder output
  - Part 3 is a fully connected layer identical to encoder part 2
  - Each part includes normalization and dropout as in the encoder

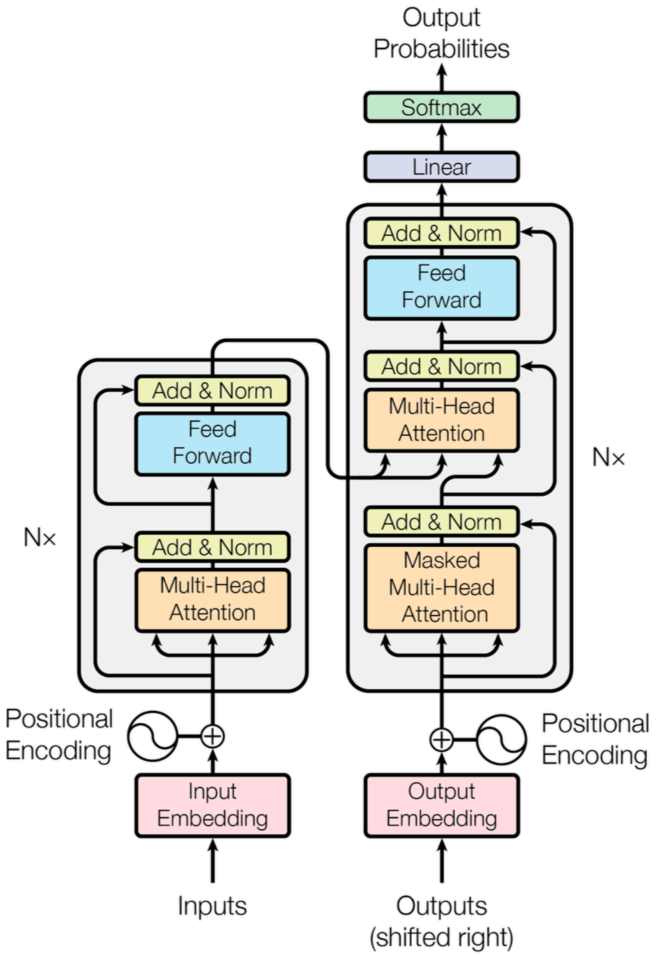
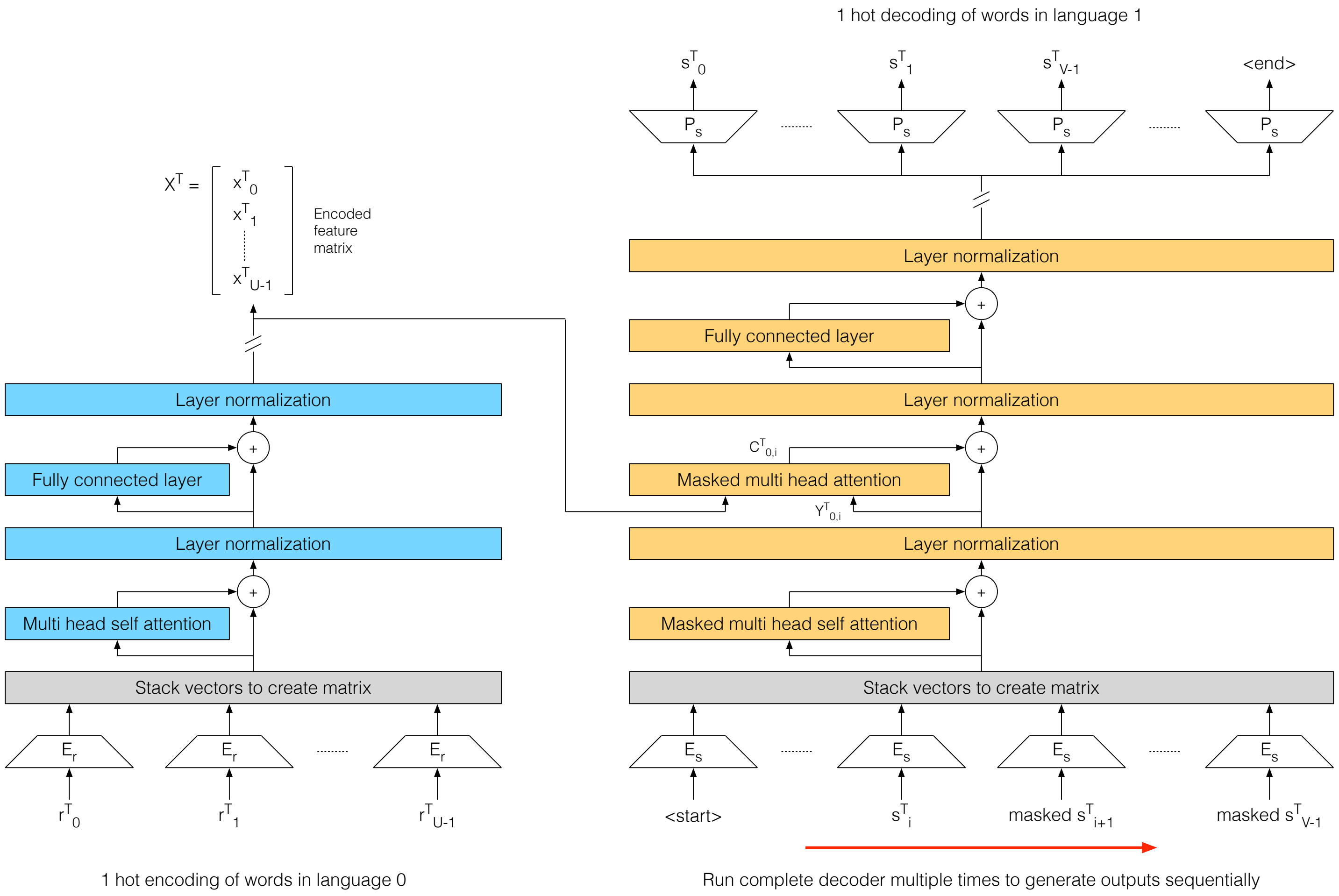


Figure from https://arxiv.org/abs/1706.03762

Self Attention Encoder – Attention – Self Attention Decoder

Encoder and decoder use self attention to allow direct mixing across feature vectors in a non sequential manner; masking is used in the decoder to generate outputs sequentially in a causal manner; as before attention allows the decoder full access to encoded features via $C^T_{l,i}$ = SoftMaxRow(($Y^T_{l,i}$ $W_{l,q}$ $W^T_{l,k}$ X) / scale) X^T $W_{l,v}$



A Few More Notes

- Can use NMT with attention and feed English and French sentences in (separately) for the same Spanish translation
  - Result is that the model encodes both languages to the same feature vector
  - The feature vector then becomes language agnostic
  - See also the slides on language agnostic sentence embeddings

- Cycle consistency
  - Map to 1 domain, should be able to go back to other domain

- Breaking the beam search curse: a study of (re-)scoring methods and stopping criteria for neural machine translation
  - https://arxiv.org/abs/1808.09582

- A survey of multilingual neural machine translation
  - https://arxiv.org/abs/1905.05395

Comparison To Speech To Text

- While both speech to text and language translation are both sequence to sequence problems there are some differences to note

- Localized monotonic alignment
  - Speech to text has a ~ localized monotonic alignment of input sound to output text
    - While attention can help with deciding exactly how big a window of the sound is useful to determine the text, it should typically be pretty localized with respect to the sound that's generating the text and the same order of input sounds is represented in output text
    - In practice this makes attention less important in speech to text than in language translation
  - Language translation doesn't have a localized monotonic alignment of input word to output word
    - If it did then something like simple word translation would potentially work
    - Attention is useful in this case as it allows the encoder and decoder to look across the full sentence representation when extracting information
    - This is needed as the data needed to extract the information is not always localized

Comparison To Speech To Text

- Input length
  - Speech input is typically much longer than output for common sound representations
    - ~ 100 frames per second of sound representation
    - Much less than 100 characters per second
    - This makes it easier to apply things like CNNs in the encoder and not worry about boundary issues (zero padding etc); in practice, pooling or down sampling is common
  - Language input and output are typically of similar length and both relatively short as compared to speech input and output
    - Boundary issues matter

References

Tutorials

- Stanford CS224n natural language processing with deep learning
  - http://web.stanford.edu/class/cs224n/
- Oxford deep NLP 2017 course
  - https://github.com/oxford-cs-deepnlp-2017/lectures
- An introduction to deep learning (lectures 2 and 3)
  - http://www.cs.toronto.edu/~ranzato/files/ranzato_deeplearn17_lec2_nlp.pdf
  - http://www.cs.toronto.edu/~ranzato/files/ranzato_deeplearn17_lec3_sequences.pdf
- Analyzing and tackling challenges in NMT
  - https://ranzato.github.io/publications/ranzato_harvard_1march18.pdf
- A primer on neural network models for natural language processing
  - https://u.cs.biu.ac.il/~yogo/nnlp.pdf
- Primer on neural network models for natural language processing
  - https://machinelearningmastery.com/primer-neural-network-models-natural-language-processing/

Tutorials

- Neural machine translation (seq2seq) tutorial
  - https://github.com/tensorflow/nmt
- Deep learning for natural language processing: tutorials with Jupyter notebooks
  - https://insights.untapt.com/deep-learning-for-natural-language-processing-tutorials-with-jupyter-notebooks-ad67f336ce3f

Data

- Conference on machine translation
  - http://statmt.org/wmt18/index.html
- Large movie review dataset
  - http://ai.stanford.edu/~amaas/data/sentiment/
- Tagged and cleaned Wikipedia (TC Wikipedia) and its ngram
  - https://nlp.cs.nyu.edu/wikipedia-data/
- The WikiText long term dependency language modeling dataset
  - https://www.salesforce.com/products/einstein/ai-research/the-wikitext-dependency-language-modeling-dataset/
- WIT3 web inventory of transcribed and translated talks
  - https://wit3.fbk.eu/mt.php?release=2016-01
- XNLI: the cross-lingual NLI corpus
  - https://github.com/facebookresearch/XNLI
- Yelp open dataset
  - https://www.yelp.com/dataset
- WordNet a lexical database for English
  - https://wordnet.princeton.edu

Word Embedding

- Word vectors and lexical semantics
  - https://github.com/oxford-cs-deepnlp-2017/lectures/blob/master/Lecture%202a-%20Word%20Level%20Semantics.pdf
- Distributed representations of words and phrases and their compositionality
  - https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf
- Efficient estimation of word representations in vector space
  - https://arxiv.org/abs/1301.3781
- Word2vec
  - https://code.google.com/archive/p/word2vec/
- Word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method
  - https://arxiv.org/abs/1402.3722
- GloVe: global vectors for word representation
  - https://nlp.stanford.edu/pubs/glove.pdf
  - https://nlp.stanford.edu/projects/glove/

Word Embedding

- Embedding word similarity with neural machine translation
    - https://arxiv.org/abs/1412.6448
- Learning structured text representations
    - https://arxiv.org/abs/1705.09207
- Non-distributional word vector representations
    - http://www.aclweb.org/anthology/P15-2076
- Deep contextualized word representations
    - https://arxiv.org/abs/1802.05365
- Linguistic regularities in continuous space word representations
    - https://www.aclweb.org/anthology/N13-1090
- What are word embeddings for text?
    - https://machinelearningmastery.com/what-are-word-embeddings/
- Word embedding benchmarks
    - https://github.com/kudkudak/word-embeddings-benchmarks

Word Embedding

- An overview of word embeddings and their connection to distributional semantic models
  - http://blog.aylien.com/overview-word-embeddings-history-word2vec-cbow-glove/
- Vector representations of words
  - https://www.tensorflow.org/tutorials/representation/word2vec
  - https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/word2vec/word2vec_basic.py
  - https://github.com/tensorflow/models/blob/master/tutorials/embedding/word2vec.py
- Word2Vec tutorial - the skip-gram model
  - http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
- Word2Vec tutorial part 2 - negative sampling
  - http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/
- Stop Using word2vec
  - https://multithreaded.stitchfix.com/blog/2017/10/18/stop-using-word2vec/

Word Embedding

- Lecture notes: part 1
  - https://cs224d.stanford.edu/lecture_notes/notes1.pdf
- Neural word embedding as implicit matrix factorization
  - https://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization.pdf
- Inducing domain-specific sentiment lexicons from unlabeled corpora
  - https://arxiv.org/abs/1606.02820
- Diachronic word embeddings reveal statistical laws of semantic change
  - https://arxiv.org/abs/1605.09096
- Linguistic regularities in continuous space word representations
  - https://www.aclweb.org/anthology/N13-1090
- How to evaluate word embeddings? On importance of data efficiency and simple supervised tasks
  - https://arxiv.org/abs/1702.02170

Visualization

- Visualizing data using t-SNE
  - http://jmlr.csail.mit.edu/papers/volume9/vandermaaten08a/vandermaaten08a.pdf
- How to use t-SNE effectively
  - https://distill.pub/2016/misread-tsne/

Language Modeling

- Recurrent neural networks and language modelling: part 1
  - https://github.com/oxford-cs-deepnlp-2017/lectures/blob/master/Lecture%203%20-%20Language%20Modelling%20and%20RNNs%20Part%201.pdf
- Recurrent neural networks and language modelling: part 2
  - https://github.com/oxford-cs-deepnlp-2017/lectures/blob/master/Lecture%204%20-%20Language%20Modelling%20and%20RNNs%20Part%202.pdf
- Pointer sentinel mixture models
  - https://arxiv.org/abs/1609.07843
- Improving neural language models with a continuous cache
  - https://arxiv.org/abs/1612.04426
- On the state of the art of evaluation in neural language models
  - https://arxiv.org/abs/1707.05589
- Regularizing and optimizing LSTM language models
  - https://arxiv.org/abs/1708.02182

Language Modeling

- Universal language model fine-tuning for text classification
  - https://arxiv.org/abs/1801.06146
- Deep contextualized word representations
  - https://arxiv.org/abs/1802.05365
- Language modeling at scale
  - https://arxiv.org/abs/1810.10045
- A neural probabilistic language model
  - http://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf
- Neural network based language models for highly inflective languages
  - http://www.fit.vutbr.cz/research/groups/speech/publi/2009/mikolov_ic2009_nnlm_4.pdf
- An estimate of an upper bound for the entropy of English
  - http://www.cs.cmu.edu/~roni/11761/PreviousYearsHandouts/gauntlet.pdf
- Notes on state of the art techniques for language modeling
  - https://www.fast.ai/2017/08/25/language-modeling-sota/

Sentence Embeddings

- NLP's ImageNet moment has arrived
  - https://thegradient.pub/nlp-imagenet/
- BERT: pre-training of deep bidirectional transformers for language understanding
  - https://arxiv.org/abs/1810.04805
- Open sourcing BERT: state-of-the-art pre-training for natural language processing
  - https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html
- TensorFlow code and pre-trained models for BERT
  - https://github.com/google-research/bert
- BERT end to end (fine-tuning + predicting) in 5 minutes with cloud TPU
  - https://colab.research.google.com/github/tensorflow/tpu/blob/master/tools/colab/bert_finetuning_with_cloud_tpus.ipynb
- The illustrated BERT, ELMo, and co. (how NLP cracked transfer learning)
  - http://jalammar.github.io/illustrated-bert/
- Demystifying BERT: a comprehensive guide to the groundbreaking NLP framework
  - https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/
- BERT
  - http://speech.ee.ntu.edu.tw/~tlkagk/courses/ML_2019/Lecture/BERT%20(v3).pdf

Sentence Embeddings

- Improving language understanding by generative pre-training
  - https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf
- Code and model for the paper "Improving language understanding by generative pre-training"
  - https://github.com/openai/finetune-transformer-lm
- Improving language understanding with unsupervised learning
  - https://openai.com/blog/language-unsupervised/
- Language models are unsupervised multitask learners
  - https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
- Code for the paper "Language models are unsupervised multitask learners"
  - https://github.com/openai/gpt-2
- Better language models and their implications
  - https://openai.com/blog/better-language-models/
- Generalized language models
  - https://lilianweng.github.io/lil-log/2019/01/31/generalized-language-models.html

Translation

- Learning phrase representations using RNN encoder-decoder for statistical machine translation
  - https://arxiv.org/abs/1406.1078
- Neural machine translation by jointly learning to align and translate
  - https://arxiv.org/abs/1409.0473
- Sequence to sequence learning with neural networks
  - https://arxiv.org/abs/1409.3215
- Addressing the rare word problem in neural machine translation
  - https://arxiv.org/abs/1410.8206
- Effective approaches to attention-based neural machine translation
  - https://arxiv.org/abs/1508.04025
- Sequence level training with recurrent neural networks
  - https://arxiv.org/abs/1511.06732
- Achieving open vocabulary neural machine translation with hybrid word-character models
  - https://arxiv.org/abs/1604.00788

Translation

- Sequence-to-sequence learning as beam-search optimization
  - https://arxiv.org/abs/1606.02960
- An actor-critic algorithm for sequence prediction
  - https://arxiv.org/abs/1607.07086
- A convolutional encoder model for neural machine translation
  - https://arxiv.org/abs/1611.02344
- Language modeling with gated convolutional networks
  - https://arxiv.org/abs/1612.08083
- Massive exploration of neural machine translation architectures
  - https://arxiv.org/abs/1703.03906 and https://github.com/google/seq2seq
- Convolutional sequence to sequence learning
  - https://arxiv.org/abs/1705.03122
- Depthwise separable convolutions for neural machine translation
  - https://arxiv.org/abs/1706.03059
- Attention is all you need
  - https://arxiv.org/abs/1706.03762

Translation

- Attentive convolution: equipping CNNs with RNN-style attention mechanisms
  - https://arxiv.org/abs/1710.00519
- Classical structured prediction losses for sequence to sequence learning
  - https://arxiv.org/abs/1711.04956
- Universal neural machine translation for extremely low resource languages
  - https://arxiv.org/abs/1802.05368
- Analyzing uncertainty in neural machine translation
  - https://arxiv.org/abs/1803.00047
- Achieving human parity on automatic Chinese to English news translation
  - https://arxiv.org/abs/1803.05567
- The best of both worlds: combining recent advances in neural machine translation
  - https://arxiv.org/abs/1804.09849
- A comparison of transformer and recurrent neural networks on multilingual neural machine translation
  - https://arxiv.org/abs/1806.06957

Translation

- Pervasive attention: 2d convolutional neural networks for sequence-to-sequence prediction
  - https://arxiv.org/abs/1808.03867
- XNLI: evaluating cross-lingual sentence representations
  - https://arxiv.org/abs/1809.05053
- Attention and augmented recurrent neural networks
  - https://distill.pub/2016/augmented-rnns/
- Introducing tf-seq2seq: an open source sequence-to-sequence framework in tensorflow
  - https://ai.googleblog.com/2017/04/introducing-tf-seq2seq-open-source.html
- How to configure an encoder-decoder model for neural machine translation
  - https://machinelearningmastery.com/configure-encoder-decoder-model-neural-machine-translation/
- Machine translation
  - https://www.microsoft.com/en-us/translator/business/machine-translation/#nnt
- Sequence to sequence models
  - http://www.cse.iitd.ernet.in/~mausam/courses/col772/spring2018/lectures/21-seq2seq.pdf
- Visualizing a neural machine translation model (mechanics of seq2seq models with attention)
  - https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

Translation

- Seq2Seq tutorial
  - https://deeppavlov.ai/examples/tutorials/04_deeppavlov_chitchat.pdf
- A neural network for machine translation, at production scale
  - https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html
- The illustrated transformer
  - https://jalammar.github.io/illustrated-transformer/
- The annotated transformer
  - http://nlp.seas.harvard.edu/2018/04/03/attention.html
- Attn: illustrated attention
  - https://towardsdatascience.com/attn-illustrated-attention-5ec4ad276ee3
- Understanding and applying self attention for NLP
  - https://www.youtube.com/watch?v=OYygPG4d9H0
- Attention is all you need attentional neural network models – Lukasz Kaiser
  - https://www.youtube.com/watch?v=rBCqOTEfxvg
- Learning phrase representations using RNN encoder–decoder for statistical machine translation
  - http://emnlp2014.org/papers/pdf/EMNLP2014179.pdf

Translation

- Attention and memory in deep learning and NLP
  - http://www.wildml.com/2016/01/attention-and-memory-in-deep-learning-and-nlp/
- New advances in natural language processing to better connect people
  - https://ai.facebook.com/blog/new-advances-in-natural-language-processing-to-better-connect-people/
- A general-purpose encoder-decoder framework for TensorFlow
  - https://github.com/google/seq2seq/
- Tensor2Tensor
  - https://github.com/tensorflow/tensor2tensor
- Facebook AI Research sequence-to-sequence toolkit written in Python
  - https://github.com/pytorch/fairseq

Translation

- BLEU: a method for automatic evaluation of machine translation
  - https://www.aclweb.org/anthology/P02-1040.pdf
- On the evaluation of machine translation systems trained with back-translation
  - https://arxiv.org/abs/1908.05204

Additional Applications

- GLUE benchmark
  - https://www.nyu.edu/projects/bowman/glue.pdf
- SuperGLUE: a stickier benchmark for general-purpose language understanding systems
  - https://arxiv.org/abs/1905.00537

- Generating Wikipedia by summarizing long sequences
  - https://arxiv.org/abs/1801.10198
- Constituency parsing with a self-attentive encoder
  - https://arxiv.org/abs/1805.01052
- Image captioning
  - https://github.com/tensorflow/models/tree/master/research/im2txt
- Introducing state of the art text classification with universal language models
  - http://nlp.fast.ai/classification/2018/05/15/introducting-ulmfit.html
  - http://nlp.fast.ai/category/classification.html