CS657A: Information Retrieval
Sequence Based Models

Arnab Bhattacharya
arnabb@cse.iitk.ac.in

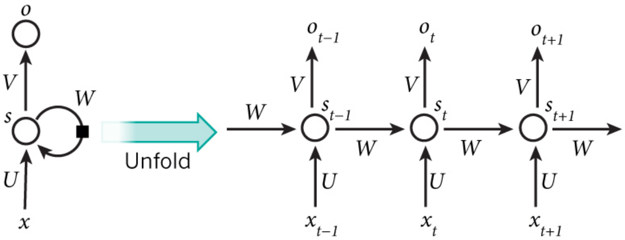
Computer Science and Engineering,
Indian Institute of Technology, Kanpur
http://web.cse.iitk.ac.in/~cs657/

2nd semester, 2021-22

Tue 1030-1145, Thu 1200-1315

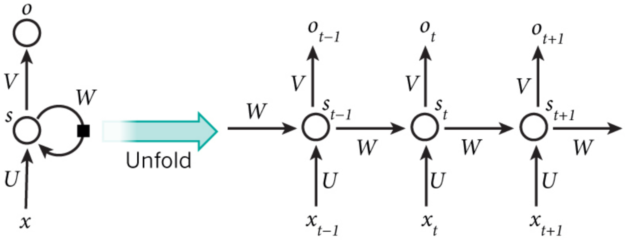
Recurrent Neural Networks

- Recurrent Neural Networks (RNNs) model recurring patterns
  - Same task is repeated for every element of a sequence
- Hidden nodes are *not* independent of each other



- Output depends on previous steps, i.e., it uses "memory"
- "Unrolling" or "unfolding" produces the layers
- If a 3-length *context* is needed, RNN is unfolded to 3 layers
- Same parameters U, V, W are shared across the layers
  - General deep networks are not constrained by this

Components of an RNN



- $\vec{x}$ at each step is the *one-hot* vector (i.e., only 1 element is on)
- $\vec{s_t}$ is "memory" as it captures everything previous

$$\vec{s_t} = f(U \cdot \vec{x_t} + W \cdot \vec{s_{t-1}} + \vec{b_s})$$

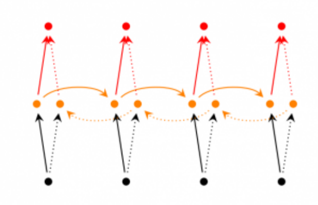
  - f is a non-linear function such as sigmoid or hyperbolic tangent
- $\vec{o_t}$ is *output* at step t

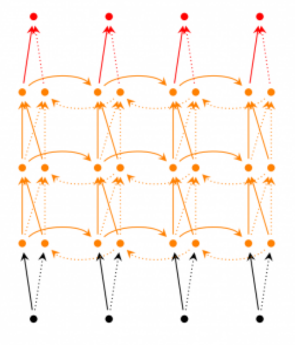
$$\vec{o_t} = g(V \cdot \vec{s_t} + \vec{b_o})$$

  - Generally, g is the *softmax* function to produce distributions

Types of RNNs

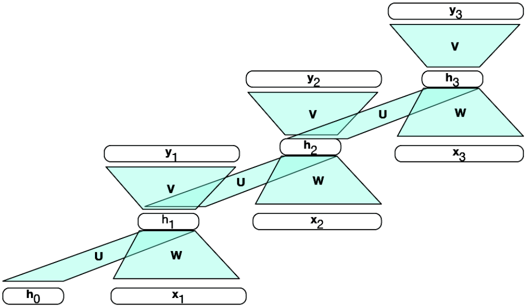
- Bi-directional RNNs use future as well as past to model present
- Deep/stacked bi-directional RNNs use multiple layers per time step





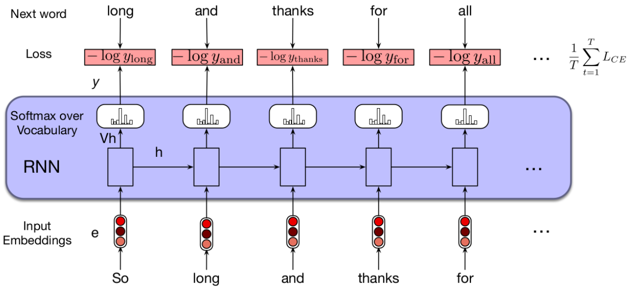
- Most famous is LSTM (long short-term memory) RNNs
  - Can use long-term memory or can ignore it
  - Instead of a simple non-linear function f at s_t, LSTM uses a complicated neural network structure

Training RNNs



- Vanilla *backpropagation* does not work since there are loops
- Unfolding removes loops
- Backpropagation is then adopted as backpropagation through time (BPTT)
- Suffers from *vanishing/exploding gradients* problem for long chains

Language Modeling using RNN

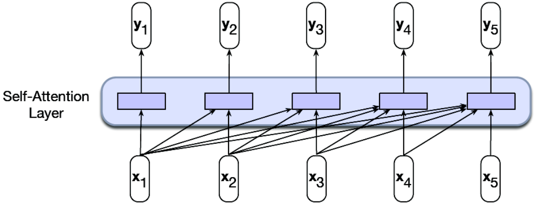


- Keep generating the next word for every time t
- Loss is average *cross-entropy loss*

$$L_{\text{cross-entropy}}(\hat{y}, y) = -\sum_i y_i \ln \hat{y}_i$$

- Word embedding vectors can be one-hot or global (e.g., Word2Vec)
- Teacher forcing sets word at $t-1$ to the *actual* word
- This is passed back to the unit at time t

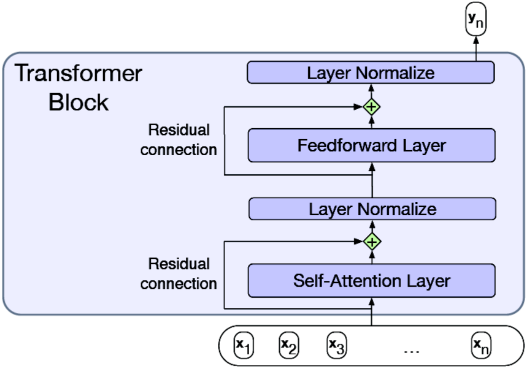
Transformers



- Transformers are *self-attention networks*
- Each unit i uses a weighted version of its previous units j as additional input
- This is called attention
- Weight depends on similarity between these two units

$$\alpha_{ij} = \text{softmax}(\vec{x}_i \cdot \vec{x}_j) \ \forall j \leq i = \frac{exp(\vec{x}_i \cdot \vec{x}_j)}{\sum_{\forall j} exp(\vec{x}_i \cdot \vec{x}_j)} \ \forall j \leq i$$

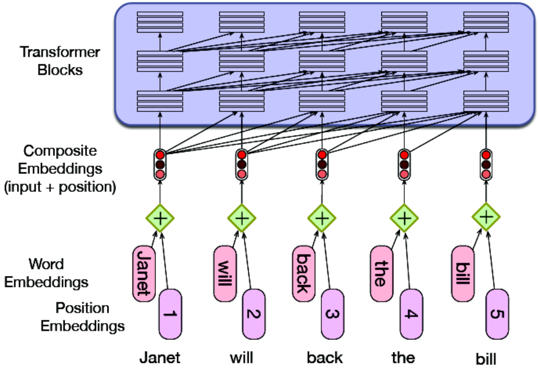
Transformer Block



- Stacked layers form a transformer block
- *Residual connections* short-circuit information by bypassing a layer
- *Layer normalization* constraints outputs to a range

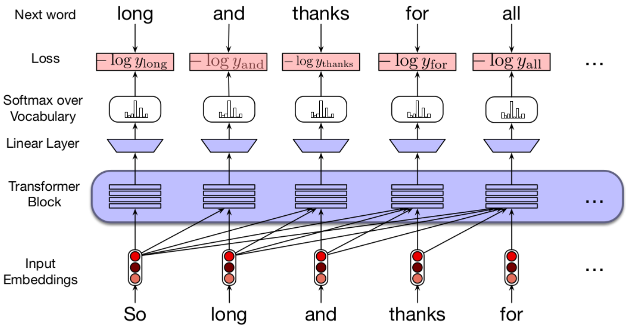
Sequence of Words

- So far, previous time steps act like a bag of words



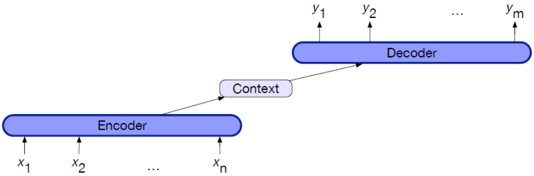
- To encode a sequence, *position vectors* are used
- Embeddings for positions are also learnt

Language Modeling using Transformers



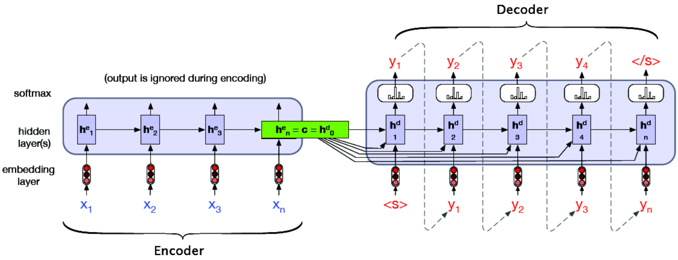
- RNN is replaced by transformer block

Encoder-Decoder Model



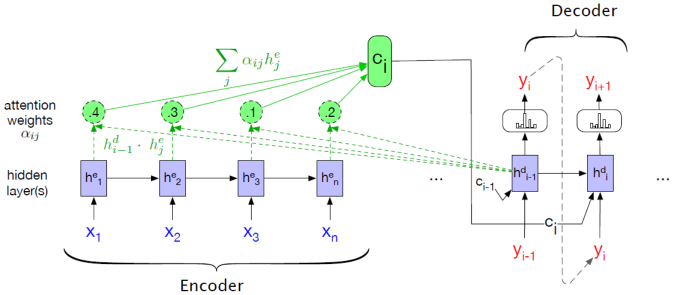
- *Encoder* accepts an input seqence and generates a sequence of contextualized representations
- *Context vector* is a function of the last contextual representation
  - This represents the entire sequence
- *Decoder* generates an arbitrary length sequence of output states

Encoder-Decoder using RNNs



- Starts with a sentence beginning marker <s>
- Keeps generating till a sentence end marker </s> is produced
- Teacher forcing is used
- Only the last encoder state matters
  - *Information bottleneck*
  - Everything about the input sequence must be captured by it
- Can use attention mechanism to resolve

Encoder-Decoder with Attention

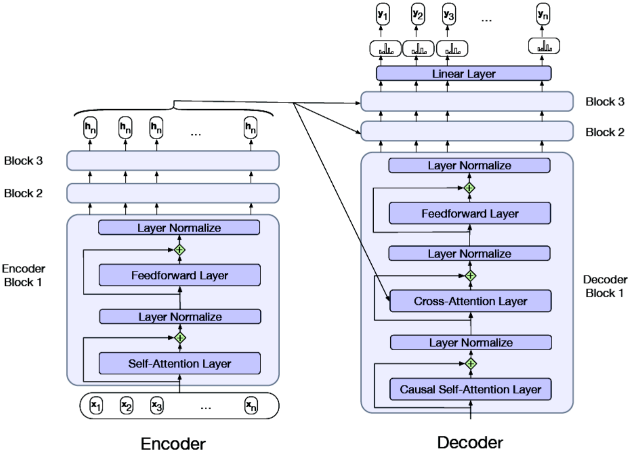


- Each decoder state gets an *attention* from every encoder state

$$\alpha_{ij} = \text{softmax}(\vec{h^d_{i-1}} \cdot \vec{h^e_j}) \; \forall j \in E$$

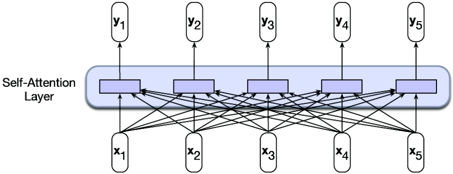
$$\vec{c_i} = \sum_{\forall j} \alpha_{ij} \vec{h^e_j}$$

Encoder-Decoder using Transformer Blocks



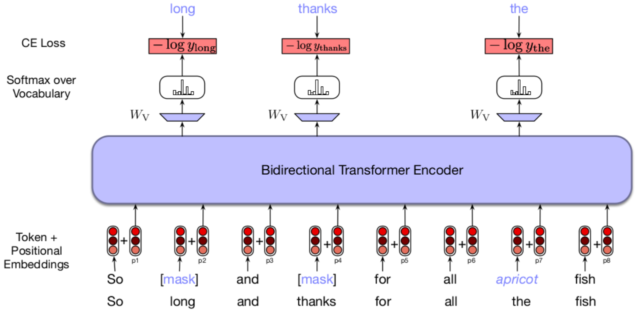
- Enormous number of parameters

Bi-directional Transformer Encoder



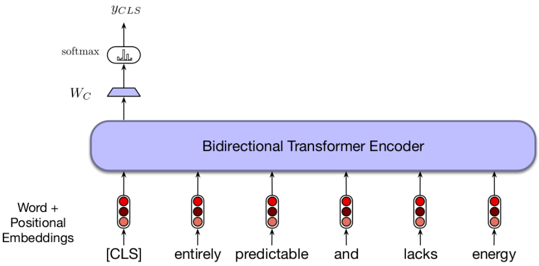
- BERT stands for *Bidirectional Encoder Representations from Transformer*
- Uses self-attention from both past and future
- Simple stacking of layers or using transformer blocks allows a time step to indirectly see itself
- Masking to resolve that

Masked Language Model



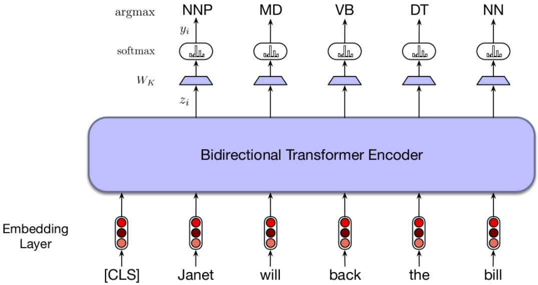
- Words are *masked*, i.e., they are replaced by special `[MASK]` tokens
- Sometimes they are replaced deliberately by *unrelated* words
- BERT uses *subword tokens* instead of actual words
- Produces contextual embeddings, i.e., embeddings of a word in context of the sequence

Using BERT



- Fine-tuning allows contextual *pre-trained* word vectors to be used for downstream IR/NLP tasks
- For sentence tasks, a special `[CLS]` token is prepended
- Vector of `[CLS]` is used for tasks with *task-specific training data*
  - Sentiment classification
  - Sentence entailment

Word Tasks using BERT



- Each individual *pre-trained* word vector is fine-tuned using a separate network for specific tasks
  - POS tagging
  - NER identification
- Since subwords may not be exactly aligned with words, a word is assigned the class to which its first subword belongs to
- Training assigns the golden class to all subwords

Discussion

- BERT is highly successful in many NLP tasks
- Requires a large number of parameters (10 crores+)
  - Subword vocabulary of size 30,000
  - Hidden layers of size 768
  - 12 layers of transformer blocks
  - 12 multi-head attention layers in each transformer block
- Hence, requires a large training corpus
- Further tasks require smaller sized training corpora
- Quality of corpus is very important since pre-trained vectors are *contextual*