Recurrent Neural Networks and Transformers

CS114B Lab 11

Kenneth Lai

April 14, 2023

Sequence Labeling

- Suppose we observe a list of words X. What are the respective parts of speech Y?
  - What is $P(Y|X)$?

Sequence Labeling

- Suppose we observe a list of words X. What are the respective parts of speech Y?
  - What is $P(Y|X)$?
- Generative approach: Hidden Markov Models

Sequence Labeling

- Suppose we observe a list of words X. What are the respective parts of speech Y?
  - What is $P(Y|X)$?
- Generative approach: Hidden Markov Models
- Discriminative approaches:
  - Conditional random fields
  - Structured perceptrons

Sequence Labeling

- Suppose we observe a list of words X. What are the respective parts of speech Y?
  - What is $P(Y|X)$?
- Generative approach: Hidden Markov Models
- Discriminative approaches:
  - Conditional random fields
  - Structured perceptrons
  - Neural networks

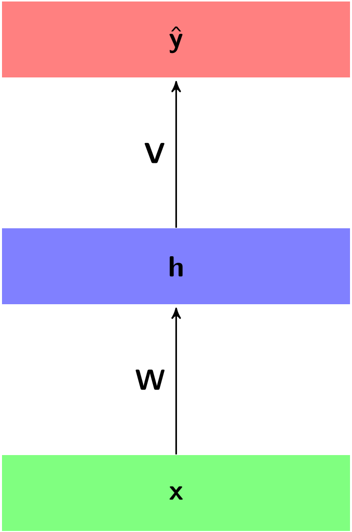
Sequence Labeling

- Discriminative approaches:
  - At each time step, use local features to compute local scores, and use the Viterbi algorithm to make predictions for the whole sentence
    - Conditional random fields
    - Structured perceptrons

Sequence Labeling

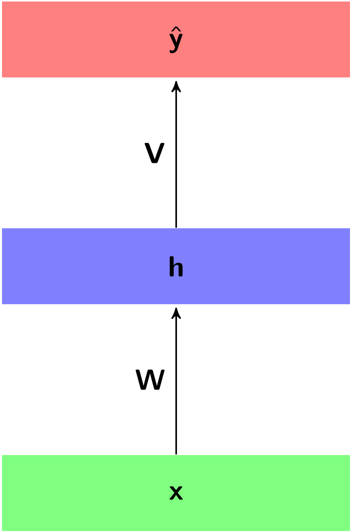
- Discriminative approaches:
  - At each time step, use local features to compute local scores, and use the Viterbi algorithm to make predictions for the whole sentence
    - Conditional random fields
    - Structured perceptrons
  - Use features from other time steps, but make independent predictions at each time step
    - Neural networks

Feedforward Neural Networks



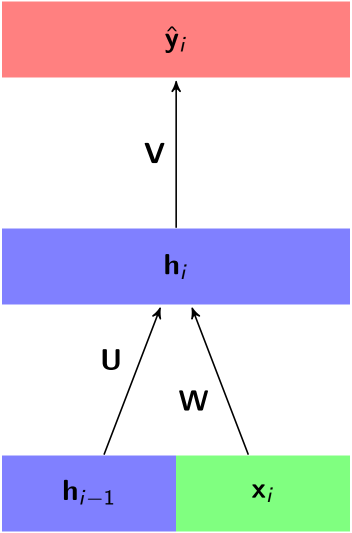
- Output layer
- Hidden layer(s)
- Input layer
- $\mathbf{h} = g(\mathbf{x}\mathbf{W})$
- $\hat{\mathbf{y}} = g'(\mathbf{h}\mathbf{V})$

Feedforward Neural Networks

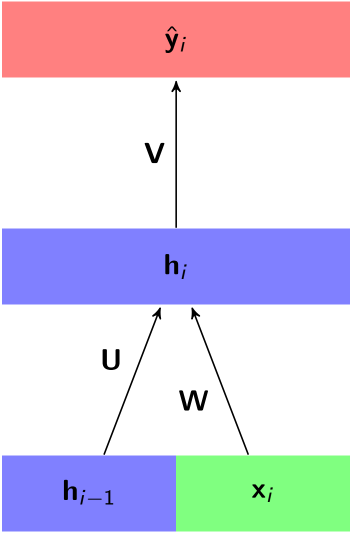


- Output layer
- Hidden layer(s)
- Input layer
- $\mathbf{h} = g(\mathbf{xW})$
- $\hat{\mathbf{y}} = g'(\mathbf{hV})$
  - We will assume that the dummy feature 1 is part of $\mathbf{x}$ and $\mathbf{h}$, and that the bias term is part of $\mathbf{W}$ and $\mathbf{V}$, etc.

Neural Networks for Sequence Labeling

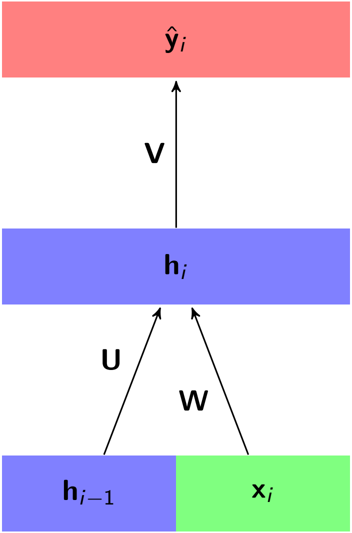


Neural Networks for Sequence Labeling



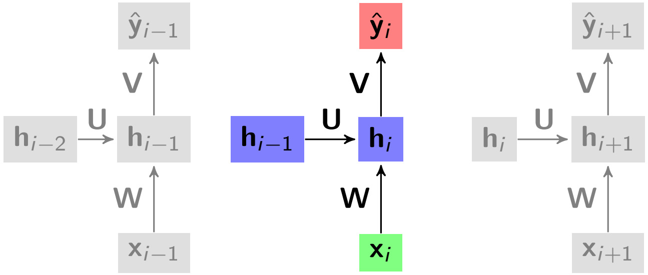
- At each time i, the input to the neural network consists of:
  - Current word (or other input) vector $\mathbf{x}_i$
  - History/(past) context vector $\mathbf{h}_{i-1}$

Neural Networks for Sequence Labeling

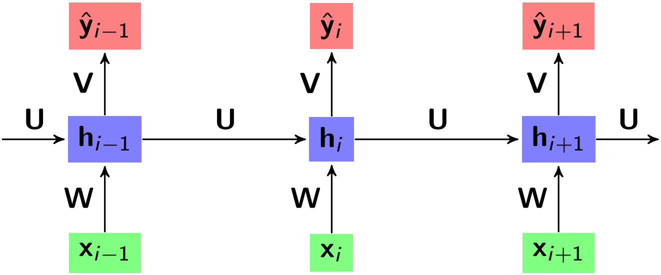


- At each time i, the input to the neural network consists of:
  - Current word (or other input) vector $\mathbf{x}_i$
  - History/(past) context vector $\mathbf{h}_{i-1}$
- $\mathbf{h}_i = g(\mathbf{x}_i \mathbf{W} + \mathbf{h}_{i-1} \mathbf{U})$

Neural Networks for Sequence Labeling

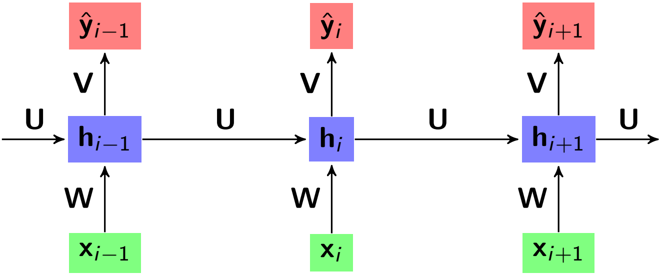


Neural Networks for Sequence Labeling



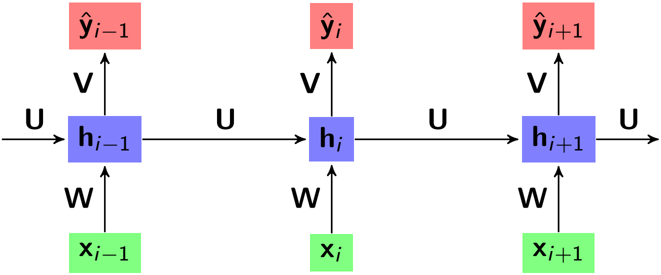
- The output of the hidden state at one time step is the history/past context input for the next time step!

Neural Networks for Sequence Labeling



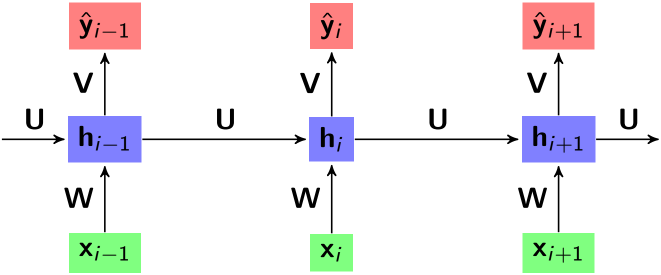
- The output of the hidden state at one time step is the history/past context input for the next time step!
- What context information is embedded in $\mathbf{h}_{i-1}$?
  - Previous word $\mathbf{x}_{i-1}$
  - Previous context $\mathbf{h}_{i-2}$

Neural Networks for Sequence Labeling



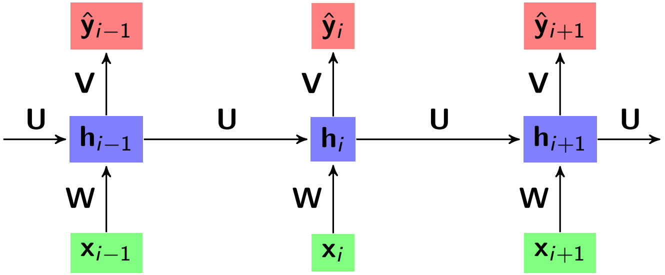
- The output of the hidden state at one time step is the history/past context input for the next time step!
- What context information is embedded in $\mathbf{h}_{i-1}$?
  - Previous word $\mathbf{x}_{i-1}$
  - Previous context $\mathbf{h}_{i-2}$
    - Previous previous word $\mathbf{x}_{i-2}$
    - Previous previous context $\mathbf{h}_{i-3}$

Neural Networks for Sequence Labeling



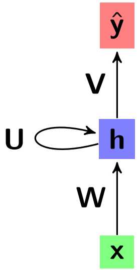
- The output of the hidden state at one time step is the history/past context input for the next time step!
- What context information is embedded in $\mathbf{h}_{i-1}$?
  - All previous words

Neural Networks for Sequence Labeling

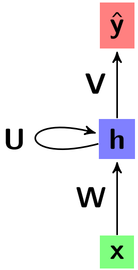


- The output of the hidden state at one time step is the history/past context input for the next time step!
- What context information is embedded in $\mathbf{h}_{i-1}$?
  - All previous words
  - What about previous parts of speech (as in HMMs, CRFs, structured perceptrons)?
    - To learn more, take CS231A in the fall!

Recurrent Neural Networks

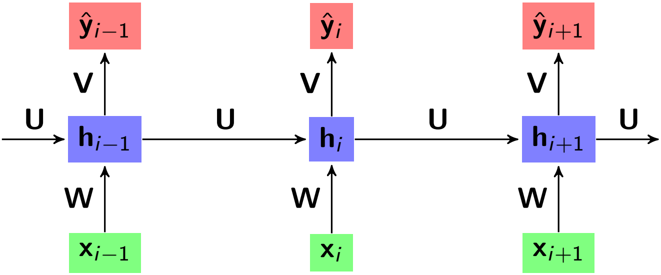


Recurrent Neural Networks

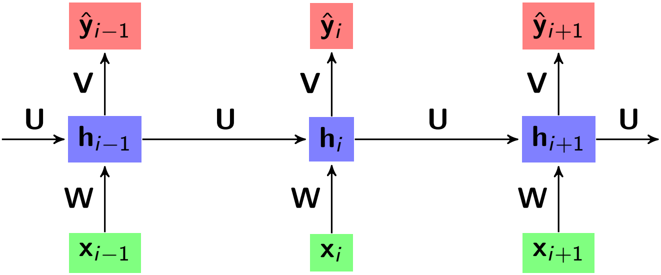


- Neural networks in which the output of a layer in one time step is input to a layer in the next time step

RNN Language Models

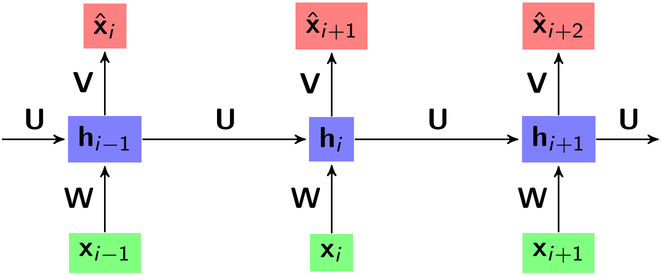


RNN Language Models



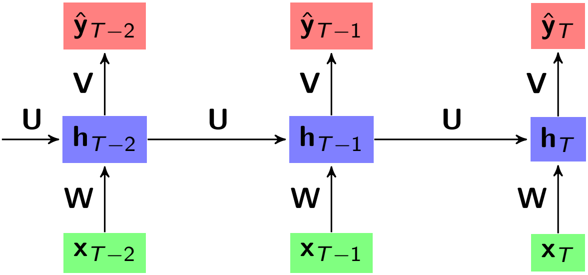
- Sequence labeling: predict current tag given current word, history
- Language modeling: predict next word given current word, history

RNN Language Models

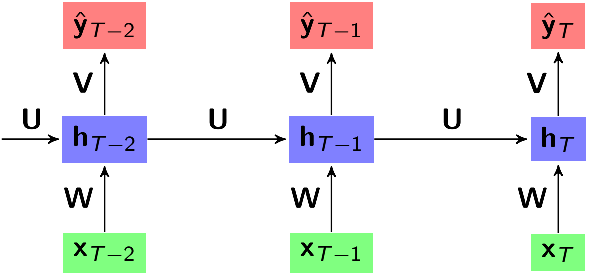


- Sequence labeling: predict current tag given current word, context
- Language modeling: predict next word given current word, context

RNNs for Text Classification

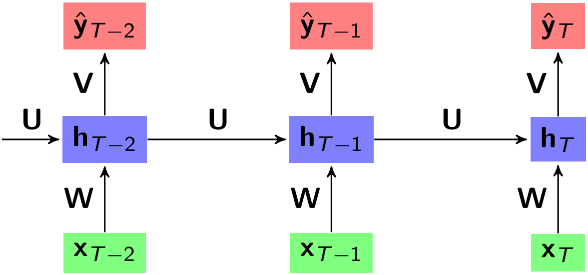


RNNs for Text Classification



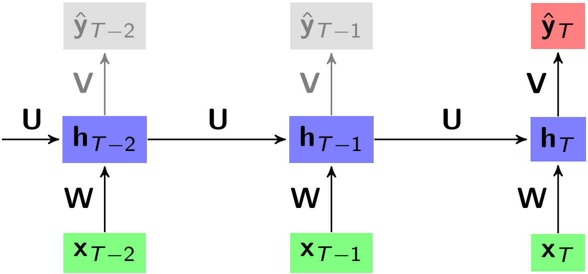
- What context information is embedded in $\mathbf{h}_T$?
  - Current word $\mathbf{x}_T$
  - Context $\mathbf{h}_{T-1}$

RNNs for Text Classification



- What context information is embedded in $\mathbf{h}_T$?
  - All words (i.e. the whole text)
- Use $\mathbf{h}_T$ to predict class $\hat{\mathbf{y}}_T$ of entire document

RNNs for Text Classification

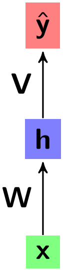


- What context information is embedded in $\mathbf{h}_T$?
  - All words (i.e. the whole text)
- Use $\mathbf{h}_T$ to predict class $\hat{\mathbf{y}}_T$ of entire document
  - Ignore other outputs

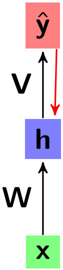
Backpropagation

- For each matrix of weights $\mathbf{W}$, starting from the output and working backwards:
  - Compute gradient $(\nabla L)^{[\mathbf{W}]}$
- For each matrix of weights $\mathbf{W}$:
  - Move in direction of negative gradient

Backpropagation

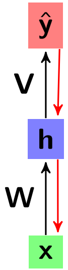


Backpropagation



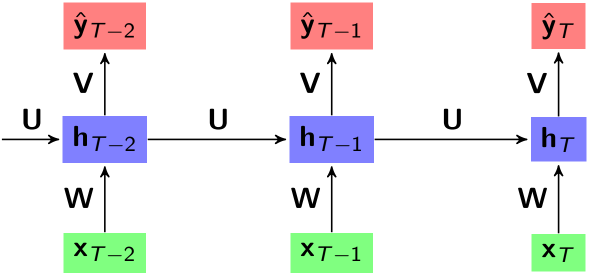
- Compute gradient $(\nabla L)^{[\mathbf{V}]}$

Backpropagation

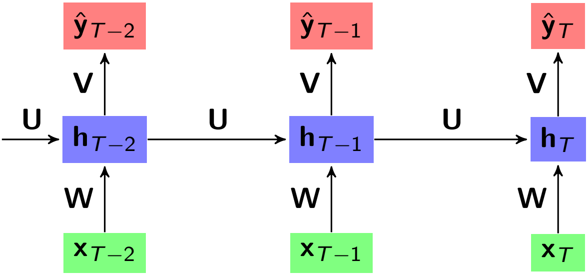


- Compute gradient $(\nabla L)^{[\mathbf{V}]}$
- Use $(\nabla L)^{[\mathbf{V}]}$ to compute gradient $(\nabla L)^{[\mathbf{W}]}$

Backpropagation Through Time

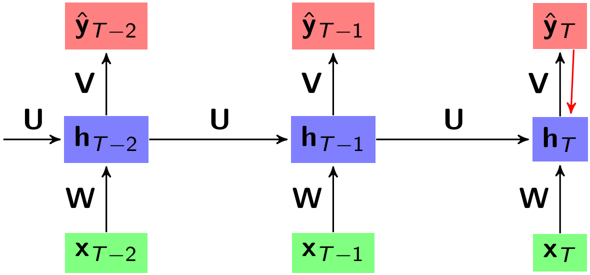


Backpropagation Through Time



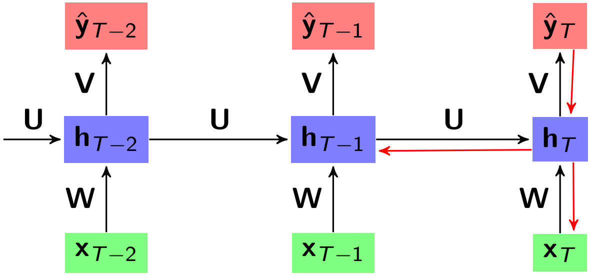
- Start at the end of the text and work backwards
  - Let $(\nabla L)_{i,j}^{[\mathbf{W}]}$ denote the part of the gradient for weight matrix $\mathbf{W}$ at time i that comes from the output at time j

Backpropagation Through Time



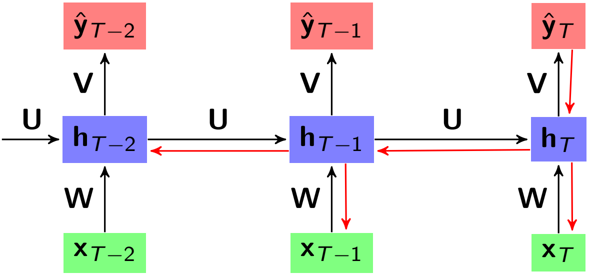
- Start at the end of the text and work backwards
  - Compute gradient $(\nabla L)^{[\mathbf{V}]}_{T,T}$

Backpropagation Through Time



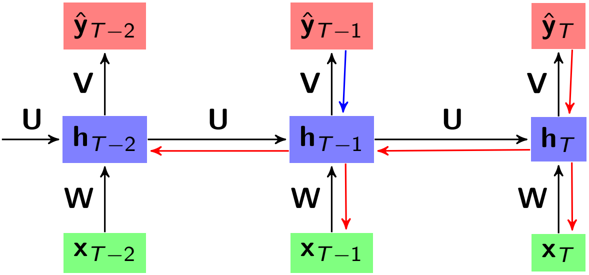
- Start at the end of the text and work backwards
  - Compute gradient $(\nabla L)^{[\mathbf{V}]}_{T,T}$
  - Use $(\nabla L)^{[\mathbf{V}]}_{T,T}$ to compute gradients $(\nabla L)^{[\mathbf{W}]}_{T,T}$ and $(\nabla L)^{[\mathbf{U}]}_{T,T}$

Backpropagation Through Time



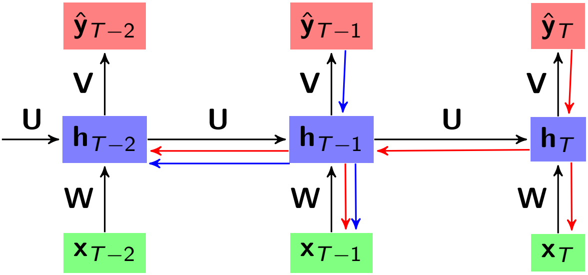
- Start at the end of the text and work backwards
  - Compute gradient $(\nabla L)^{[\mathbf{V}]}_{T,T}$
  - Use $(\nabla L)^{[\mathbf{V}]}_{T,T}$ to compute gradients $(\nabla L)^{[\mathbf{W}]}_{T,T}$ and $(\nabla L)^{[\mathbf{U}]}_{T,T}$
  - Use $(\nabla L)^{[\mathbf{U}]}_{T,T}$ to compute gradients $(\nabla L)^{[\mathbf{W}]}_{T-1,T}$ and $(\nabla L)^{[\mathbf{U}]}_{T-1,T}$
  - etc.

Backpropagation Through Time



- Start at the end of the text and work backwards
  - Compute gradient $(\nabla L)^{[\mathbf{V}]}_{T-1,T-1}$

Backpropagation Through Time



- Start at the end of the text and work backwards
  - Compute gradient $(\nabla L)^{[\mathbf{V}]}_{T-1,T-1}$
  - Use $(\nabla L)^{[\mathbf{V}]}_{T-1,T-1}$ to compute gradients $(\nabla L)^{[\mathbf{W}]}_{T-1,T-1}$ and $(\nabla L)^{[\mathbf{U}]}_{T-1,T-1}$
  - etc.

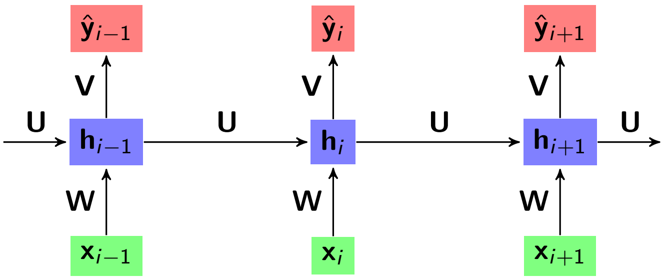
Backpropagation Through Time

- The overall gradient for a weight matrix $\mathbf{W}$ is the sum of the gradients at each time i from each output $\hat{\mathbf{y}}_j$
  - $(\nabla L)^{[\mathbf{W}]} = \sum_{j=1}^{T} \sum_{i=1}^{j} (\nabla L)_{i,j}^{[\mathbf{W}]}$

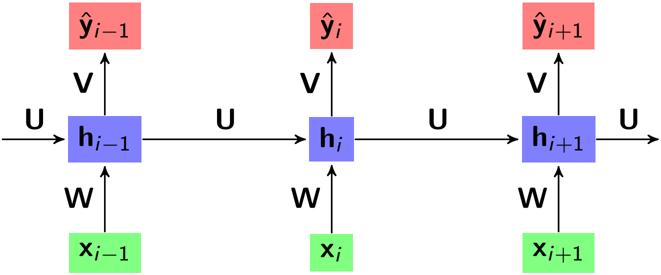
Backpropagation Through Time

- The overall gradient for a weight matrix $\mathbf{W}$ is the sum of the gradients at each time i from each output $\hat{\mathbf{y}}_j$
  - $(\nabla L)^{[\mathbf{W}]} = \sum_{j=1}^{T} \sum_{i=1}^{j} (\nabla L)_{i,j}^{[\mathbf{W}]}$
- Then move in direction of negative gradient

Recurrent Neural Networks



Recurrent Neural Networks

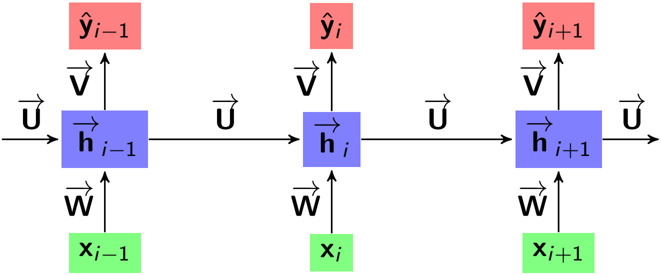


- Output $\hat{\mathbf{y}}_i$ depends on hidden state $\mathbf{h}_i$ (i.e. current word $\mathbf{x}_i$ and history/(past) context $\mathbf{h}_{i-1}$)
- What about future context?

Bidirectional RNNs

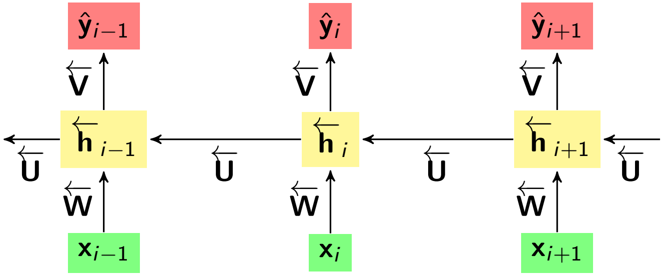
- Idea: Train two RNNs: passing the input into one forward and one backward
- Output $\hat{\mathbf{y}}_i$ depends on forward hidden state $\overrightarrow{\mathbf{h}}_i$ and backward hidden state $\overleftarrow{\mathbf{h}}_i$

Forward RNN

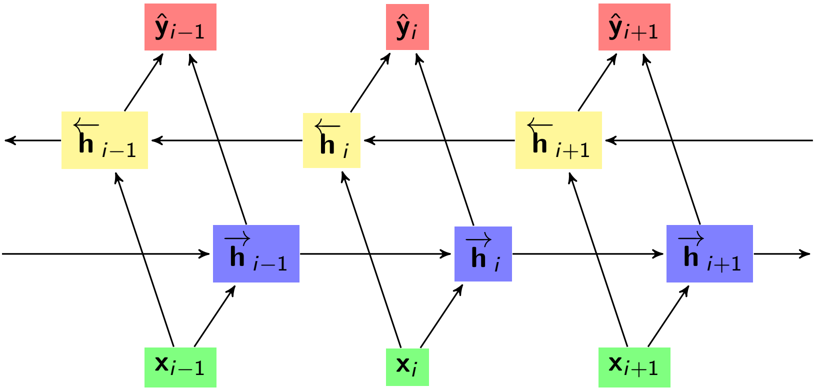


$\hat{\mathbf{y}}_{i-1}$
$\hat{\mathbf{y}}_i$
$\hat{\mathbf{y}}_{i+1}$
$\overrightarrow{\mathbf{V}}$
$\overrightarrow{\mathbf{V}}$
$\overrightarrow{\mathbf{V}}$
$\overrightarrow{\mathbf{U}}$
$\overrightarrow{\mathbf{U}}$
$\overrightarrow{\mathbf{U}}$
$\overrightarrow{\mathbf{U}}$
$\overrightarrow{\mathbf{h}}_{i-1}$
$\overrightarrow{\mathbf{h}}_i$
$\overrightarrow{\mathbf{h}}_{i+1}$
$\overrightarrow{\mathbf{W}}$
$\overrightarrow{\mathbf{W}}$
$\overrightarrow{\mathbf{W}}$
$\mathbf{x}_{i-1}$
$\mathbf{x}_i$
$\mathbf{x}_{i+1}$

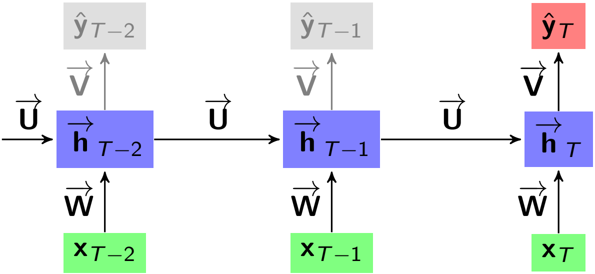
Backward RNN



Bidirectional RNN

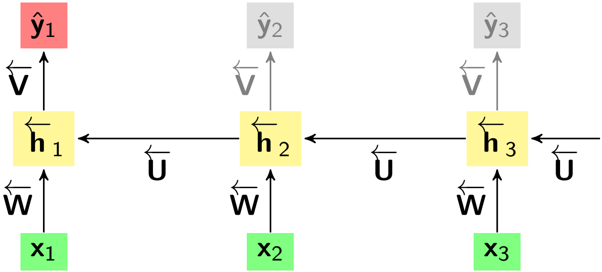


Bidirectional RNNs for Text Classification



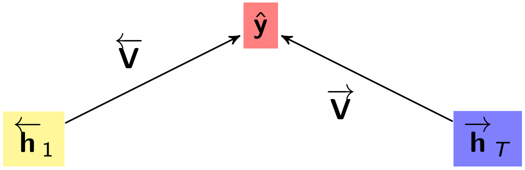
- $\overrightarrow{\mathbf{h}}_T$ encodes the whole text
  - Use $\overrightarrow{\mathbf{h}}_T$ to predict class $\hat{\mathbf{y}}_T$ of entire document

Bidirectional RNNs for Text Classification



- $\overrightarrow{\mathbf{h}}_T$ encodes the whole text
  - Use $\overrightarrow{\mathbf{h}}_T$ to predict class $\hat{\mathbf{y}}_T$ of entire document
- $\overleftarrow{\mathbf{h}}_1$ also encodes the whole text
  - Use $\overleftarrow{\mathbf{h}}_1$ to predict class $\hat{\mathbf{y}}_1$ of entire document

Bidirectional RNNs for Text Classification



- Use $\overrightarrow{\mathbf{h}}_T$ and $\overleftarrow{\mathbf{h}}_1$ to predict class $\hat{\mathbf{y}}$ of entire document

Context and Long-Distance Dependencies

- $\mathbf{h}_{i-1}$ encodes the (past, in a forward RNN) context $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$
  - But mostly $\mathbf{x}_{i-1}$, less $\mathbf{x}_{i-2}$, even less $\mathbf{x}_{i-3}$, ..., very little $\mathbf{x}_1$
- Context is local

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline (was/were) cancelling (was/were) full.

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling (was/were) full.
  - The context for “was” is mostly “airline”

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling were full.
  - The context for “was” is mostly “airline”
  - The context for “were” is mostly “cancelling”, “was”, “airline”
    - Very little “flights”

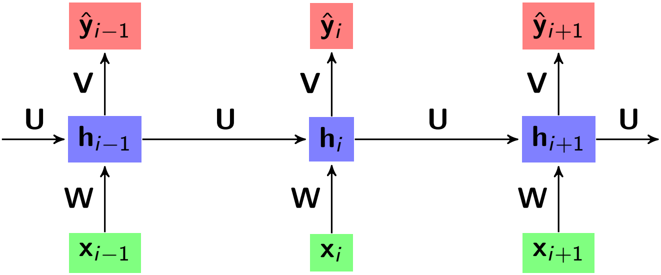
Context and Long-Distance Dependencies

- Two approaches to handling long-distance dependencies:
  - Memory-based (e.g. long short-term)
    -  does this

Context and Long-Distance Dependencies

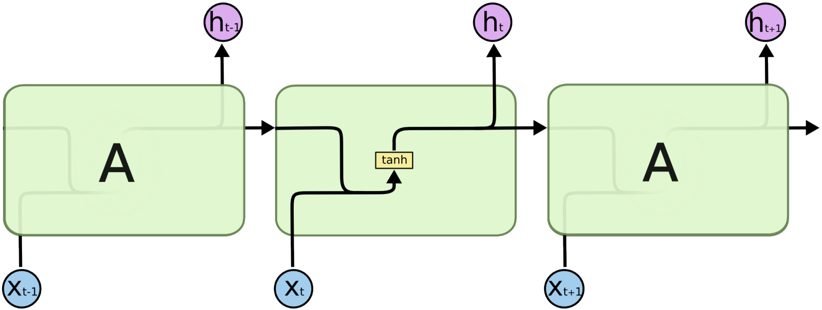
- Two approaches to handling long-distance dependencies:
  - Memory-based (e.g. long short-term)
    -  does this
  - Attention-based
    - At each time step, the model explicitly computes which other words to pay attention to
    -  does this

Simple RNN



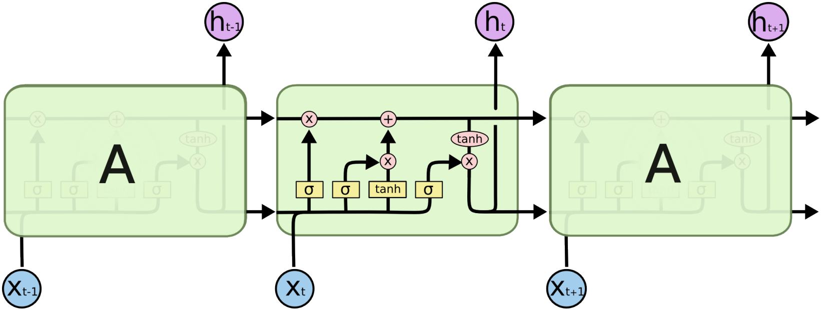
$\hat{\mathbf{y}}_{i-1}$
$\hat{\mathbf{y}}_i$
$\hat{\mathbf{y}}_{i+1}$
V
V
V
U
$\mathbf{h}_{i-1}$
U
$\mathbf{h}_i$
U
$\mathbf{h}_{i+1}$
U
W
W
W
$\mathbf{x}_{i-1}$
$\mathbf{x}_i$
$\mathbf{x}_{i+1}$

Simple RNN



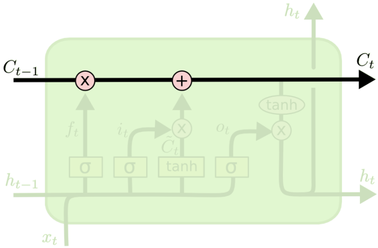
Source

Long Short-Term Memory



Source

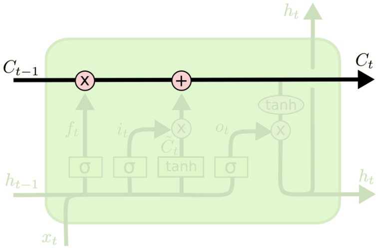
Long Short-Term Memory



Source

- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates
    - Each gate contains one or two neural network layers

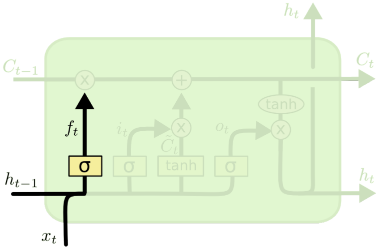
Long Short-Term Memory



Source

- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates
    - Each gate contains one or two neural network layers
  - State persists across time
    - May remember information from long ago
    - Gradients for memory don't decay with time

Forget Gate

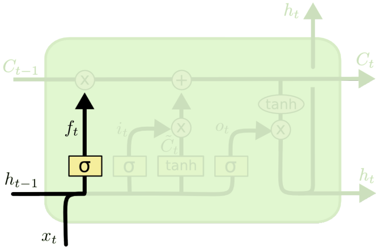


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Source

- Neural network layer with logistic activation function

Forget Gate

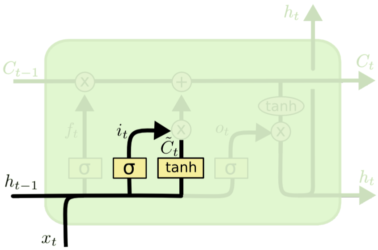


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Source

- Neural network layer with logistic activation function
- Element-wise multiplication of forget gate output with memory state
    - Mask: What parts of memory to forget/remember?

Input Gate



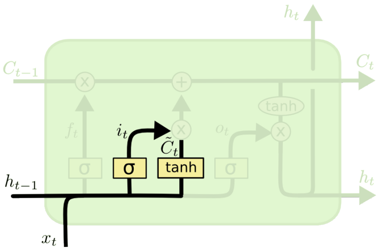
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Source

- Two parts
  1. Candidate choice
     - Logistic activation function
     - What parts of memory to update?

Input Gate

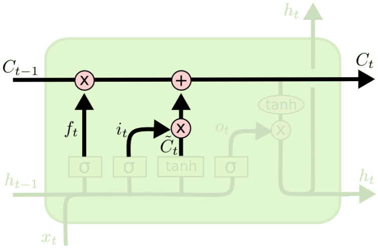


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Source

- Two parts
  1. Candidate choice
     - Logistic activation function
     - What parts of memory to update?
  2. Candidate values
     - Tanh activation function
     - How much to update them by?

Input Gate

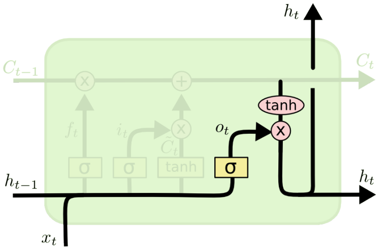


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Source

- Element-wise multiplication of two outputs
- Then element-wise addition with memory state

Output Gate

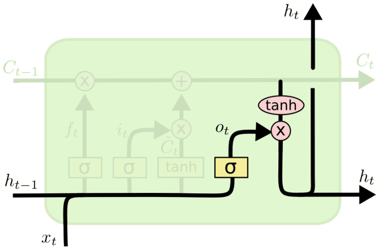


$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Source

- Logistic activation function
  - What parts of memory to output?

Output Gate



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Source

- Logistic activation function
  - What parts of memory to output?
- Element-wise multiplication with tanh of memory state
  - This is the "hidden layer output" that gets passed on to the output layer/next time step

Context and Long-Distance Dependencies

- Two approaches to handling long-distance dependencies:
  - Memory-based (e.g. long short-term)
    -  does this
  - Attention-based
    - At each time step, the model explicitly computes which other words to pay attention to
    -  does this

Attention

- Scaled dot-product self-attention

Attention

- Scaled dot-product self-attention
  - Scaled dot-product: how to compute the relevance of the other words
  - Self-attention: paying attention to the input sequence itself (rather than some output sequence)

Attention

- Input: pre-trained word vectors (e.g. from word2vec)

Attention

- Input: pre-trained word vectors (e.g. from word2vec)

1. Compute query, key, and value vectors for each input vector
    - Matrix multiplication

Attention

- For each word i:
  2. Compute the dot product of query q_i with key k_j for each word j

Attention

- For each word i:
  2. Compute the dot product of query q_i with key k_j for each word j
  3. Scale the dot product (e.g. Vaswani et al. (2017) divide by 8)
     - Leads to more stable gradients

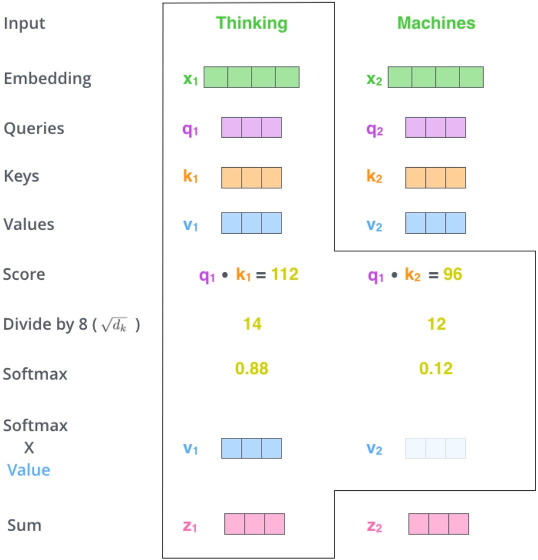
Attention

- For each word i:
  2. Compute the dot product of query q_i with key k_j for each word j
  3. Scale the dot product (e.g. Vaswani et al. (2017) divide by 8)
     - Leads to more stable gradients
  4. Softmax
     - Numbers $\rightarrow$ probabilities (or more generally, numbers between 0 and 1, that add up to 1)

Attention

- For each word i:
  2. Compute the dot product of query q_i with key k_j for each word j
  3. Scale the dot product (e.g. Vaswani et al. (2017) divide by 8)
     - Leads to more stable gradients
  4. Softmax
     - Numbers $\rightarrow$ probabilities (or more generally, numbers between 0 and 1, that add up to 1)
  5. Compute the weighted sum of values v_j for each word j
     - Weights $=$ softmax output from previous step

Attention



Source

Attention

- Output: weighted sum of value vectors (modulo some more advanced topics)
  - Multi-head attention
  - Positional encodings
  - Residual connections
  - Layer normalization

Attention

- Output: weighted sum of value vectors (modulo some more advanced topics)
  - Multi-head attention
  - Positional encodings
  - Residual connections
  - Layer normalization
- See Jay Alammar's The Illustrated Transformer for more details!

Transformers

- "Attention Is All You Need" (Vaswani et al. 2017)
- No recurrence, relies entirely on attention (and feedforward layers) to capture global dependencies

Transformers

- "Attention Is All You Need" (Vaswani et al. 2017)
- No recurrence, relies entirely on attention (and feedforward layers) to capture global dependencies
  - Recurrent neural networks are inherently sequential, processing one word at a time
  - Transformers are more parallel, looking at the entire sequence at once

Transformers

- "Attention Is All You Need" (Vaswani et al. 2017)
- No recurrence, relies entirely on attention (and feedforward layers) to capture global dependencies
  - Recurrent neural networks are inherently sequential, processing one word at a time
  - Transformers are more parallel, looking at the entire sequence at once
    - More efficient, especially on GPUs
    - Also scores better on many NLP tasks

Transformers

- Three "building blocks" of practical transformer models (Wolf et al. 2020):

Transformers

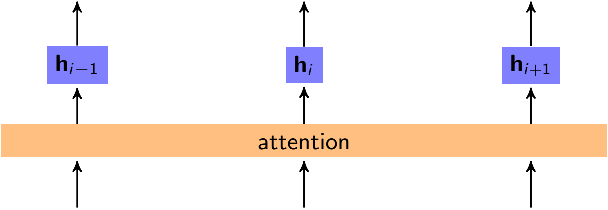
- Three "building blocks" of practical transformer models (Wolf et al. 2020):
  1. Tokenizer
     - What is the input to the transformer?
     - Examples: words, WordPieces, characters, etc.

Transformers

- Three "building blocks" of practical transformer models (Wolf et al. 2020):
  2. Transformer
     - The model itself (minus the output)

Transformers

- Three "building blocks" of practical transformer models (Wolf et al. 2020):
  2. Transformer
     - The model itself (minus the output)
     -  : 12-24 encoder layers
       - Encoder layer = (shared) attention layer + (individual) feedforward layers



($\times$ 12-24)

Transformers

- Three "building blocks" of practical transformer models (Wolf et al. 2020):
  3. Head
     - What is the output of the transformer?

Transformers

- Three "building blocks" of practical transformer models (Wolf et al. 2020):
  3. Head
     - What is the output of the transformer?
     - Models are pre-trained using one or more heads, but you can then fine-tune a model using a different head (task)

Transformers

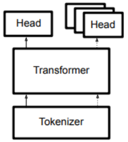
- Three "building blocks" of practical transformer models (Wolf et al. 2020):
  3. Head
     - What is the output of the transformer?
     - Models are pre-trained using one or more heads, but you can then fine-tune a model using a different head (task)
     -  : Masked LM (Cloze) and NSP (Next Sentence Prediction)
       - Masked LM: mask 15% of input words at random, predict masked words
       - NSP: given sentences A and B, does B follow A?

Transformers

- Table from Wolf et al. (2020)

Name	Heads Input	Heads Output	Tasks	Ex. Datasets
Language Modeling	$x_{1:n-1}$	$x_n \in \mathcal{V}$	Generation	WikiText-103
Sequence Classification	$x_{1:N}$	$y \in \mathcal{C}$	Classification, Sentiment Analysis	GLUE, SST, MNLI
Question Answering	$x_{1:M}, x_{M:N}$	y span $[1:N]$	QA, Reading Comprehension	SQuAD, Natural Questions
Token Classification	$x_{1:N}$	$y_{1:N} \in \mathcal{C}^N$	NER, Tagging	OntoNotes, WNUT
Multiple Choice	$x_{1:N}, \mathcal{X}$	$y \in \mathcal{X}$	Text Selection	SWAG, ARC
Masked LM	$x_{1:N \setminus n}$	$x_n \in \mathcal{V}$	Pretraining	Wikitext, C4
Conditional Generation	$x_{1:N}$	$y_{1:M} \in \mathcal{V}^M$	Translation, Summarization	WMT, IWSLT, CNN/DM, XSum

Transformers	
Masked $[x_{1:N\setminus n} \Rightarrow x_n]$	
BERT	(Devlin et al., 2018)
RoBERTa	(Liu et al., 2019a)
Autoregressive $[x_{1:n-1} \Rightarrow x_n]$	
GPT / GPT-2	(Radford et al., 2019)
Trans-XL	(Dai et al., 2019)
XLNet	(Yang et al., 2019)
Seq-to-Seq $[\sim x_{1:N} \Rightarrow x_{1:N}]$	
BART	(Lewis et al., 2019)
T5	(Raffel et al., 2019)
MarianMT	(J.-Dowmunt et al., 2018)
Specialty: Multimodal	
MMBT	(Kiela et al., 2019)
Specialty: Long-Distance	
Reformer	(Kitaev et al., 2020)
Longformer	(Beltagy et al., 2020)
Specialty: Efficient	
ALBERT	(Lan et al., 2019)
Electra	(Clark et al., 2020)
DistilBERT	(Sanh et al., 2019)
Specialty: Multilingual	
XLM/RoBERTa	(Lample and Conneau, 2019b)



Tokenizers	
Name	Ex. Uses
Character-Level BPE	NMT, GPT
Byte-Level BPE	GPT-2
WordPiece	BERT
SentencePiece	XLNet
Unigram	LM
Character	Reformer
Custom	Bio-Chem

Figure 2: The *Transformers* library. **(Diagram-Right)** Each model is made up of a Tokenizer, Transformer, and Head. The model is pretrained with a fixed head and can then be further fine-tuned with alternate heads for different tasks. **(Bottom)** Each model uses a specific Tokenizer either implemented in Python or in Rust. These often differ in small details, but need to be in sync with pretraining. **(Left)** Transformer architectures specialized for different tasks, e.g. understanding versus generation, or for specific use-cases, e.g. speed, image+text. **(Top)** heads allow a Transformer to be used for different tasks. Here we assume the input token sequence is $x_{1:N}$ from a vocabulary $\mathcal{V}$, and y represents different possible outputs, possibly from a class set $\mathcal{C}$. Example datasets represent a small subset of example code distributed with the library.