Recurrent Neural Networks and Transformers

Shan-Hung Wu
shwu@cs.nthu.edu.tw

Department of Computer Science,
National Tsing Hua University, Taiwan

Machine Learning

Outline

1. **RNNs**
   - Vanilla RNNs
   - Design Alternatives
2. **RNN Training**
   - Backprop through Time (BPTT)
   - Optimization Techniques
   - Optimization-Friendly Models & LSTM
   - Parallelism & Teacher Forcing
3. **RNNs with Attention Mechanism**
   - Attention for Image Captioning
   - Attention for Neural Machine Translation (NMT)
4. **Transformers**
   - Attention is All You Need
   - Pretrained Language Models
   - More Applications
5. **Subword Tokenization**

Outline

Outline

Sequential Data

- So far, we assume that data points $(\boldsymbol{x},\boldsymbol{y})$'s in a dataset are i.i.d

Sequential Data

- So far, we assume that data points $(\boldsymbol{x},\boldsymbol{y})$'s in a dataset are i.i.d
- Does ***not*** hold in many applications

Sequential Data

- So far, we assume that data points $(\boldsymbol{x},\boldsymbol{y})$'s in a dataset are i.i.d
- Does ***not*** hold in many applications
- ***Sequential data***: data points come in order and successive points may be dependent, e.g.,
    - Letters in a word
    - Words in a sentence/document
    - Phonemes in a spoken word utterance
    - Page clicks in a Web session
    - Frames in a video, etc.

Sequential Data

- So far, we assume that data points $(\boldsymbol{x},\boldsymbol{y})$'s in a dataset are i.i.d
- Does ***not*** hold in many applications
- ***Sequential data***: data points come in order and successive points may be dependent, e.g.,
  - Letters in a word
  - Words in a sentence/document
  - Phonemes in a spoken word utterance
  - Page clicks in a Web session
  - Frames in a video, etc.
- Dataset: $\mathbf{X} = \{\boldsymbol{X}^{(n)}\}_n \in \mathbb{R}^{N\times(D,K)\times T}$

Sequential Data

- So far, we assume that data points $(\boldsymbol{x},\boldsymbol{y})$'s in a dataset are i.i.d
- Does ***not*** hold in many applications
- ***Sequential data***: data points come in order and successive points may be dependent, e.g.,
  - Letters in a word
  - Words in a sentence/document
  - Phonemes in a spoken word utterance
  - Page clicks in a Web session
  - Frames in a video, etc.
- Dataset: $\mathbf{X} = \{\boldsymbol{X}^{(n)}\}_n \in \mathbb{R}^{N\times(D,K)\times T}$
  - $\boldsymbol{X}^{(n)} = \{(\boldsymbol{x}^{(n,t)},\boldsymbol{y}^{(n,t)})\}_t$ a ***sequence***, where the superscript n can be omitted for simplicity

Sequential Data

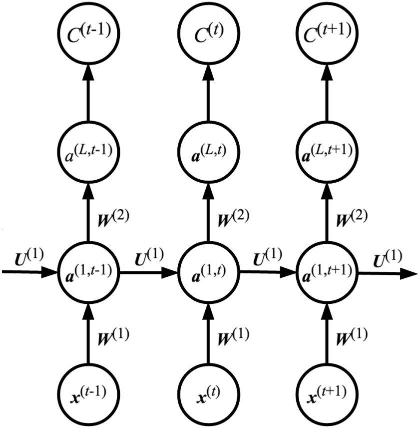
- So far, we assume that data points $(\boldsymbol{x},\boldsymbol{y})$'s in a dataset are i.i.d
- Does ***not*** hold in many applications
- ***Sequential data***: data points come in order and successive points may be dependent, e.g.,
  - Letters in a word
  - Words in a sentence/document
  - Phonemes in a spoken word utterance
  - Page clicks in a Web session
  - Frames in a video, etc.
- Dataset: $\mathbf{X} = \{\boldsymbol{X}^{(n)}\}_n \in \mathbb{R}^{N\times(D,K)\times T}$
  - $\boldsymbol{X}^{(n)} = \{(\boldsymbol{x}^{(n,t)},\boldsymbol{y}^{(n,t)})\}_t$ a ***sequence***, where the superscript n can be omitted for simplicity
  - T is call the ***horizon*** and may be different between $\boldsymbol{x}^{(n)}$ and $\boldsymbol{y}^{(n)}$ and across data points n's

Sequence Modeling I

- How to model sequential data?

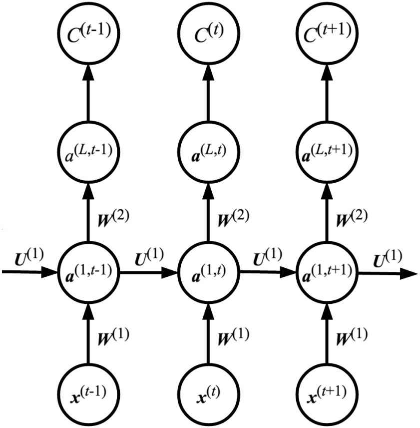
Sequence Modeling I

- How to model sequential data?
- ***Recurrent neural networks*** (vanilla RNNs):



Sequence Modeling I

- How to model sequential data?
- ***Recurrent neural networks*** (vanilla RNNs):
- $\mathbf{y}^{(t)}$ depends on $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t)}$

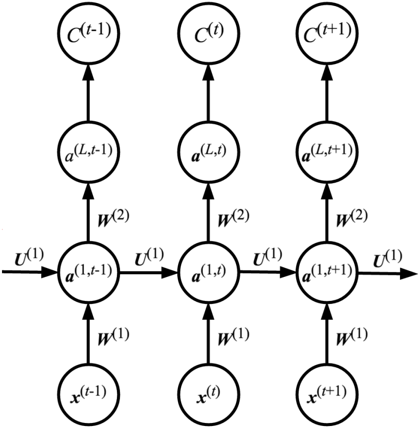


Sequence Modeling I

- How to model sequential data?
- ***Recurrent neural networks*** (vanilla RNNs):
- $\mathbf{y}^{(t)}$ depends on $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t)}$
- Output $\mathbf{a}^{(L,t)}$ depends on hidden activations:

$$\begin{aligned}\mathbf{a}^{(k,t)} &= \mathrm{act}(\mathbf{z}^{(k,t)}) \\ &= \mathrm{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})\end{aligned}$$

  - Bias term omitted

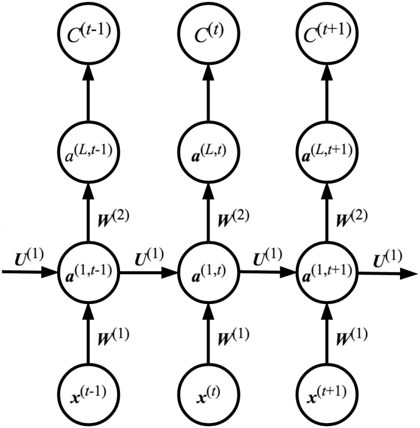


Sequence Modeling I

- How to model sequential data?
- ***Recurrent neural networks*** (vanilla RNNs):
- $\mathbf{y}^{(t)}$ depends on $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t)}$
- Output $\mathbf{a}^{(L,t)}$ depends on hidden activations:

$$\begin{aligned}\mathbf{a}^{(k,t)} &= \mathrm{act}(\mathbf{z}^{(k,t)}) \\ &= \mathrm{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})\end{aligned}$$

  - Bias term omitted
- $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \cdots, \mathbf{x}^{(1)}$
  - Earlier points are less important

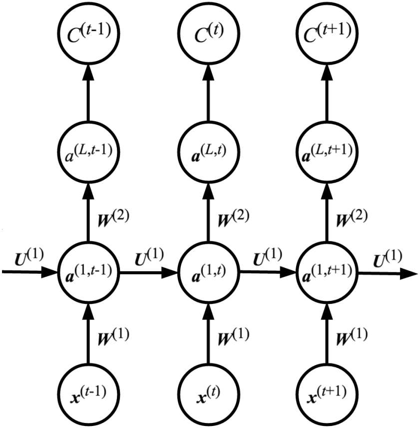


Sequence Modeling I

- How to model sequential data?
- ***Recurrent neural networks*** (vanilla RNNs):
- $\mathbf{y}^{(t)}$ depends on $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t)}$
- Output $\mathbf{a}^{(L,t)}$ depends on hidden activations:

$$\begin{aligned}\mathbf{a}^{(k,t)} &= \mathrm{act}(\mathbf{z}^{(k,t)}) \\ &= \mathrm{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})\end{aligned}$$

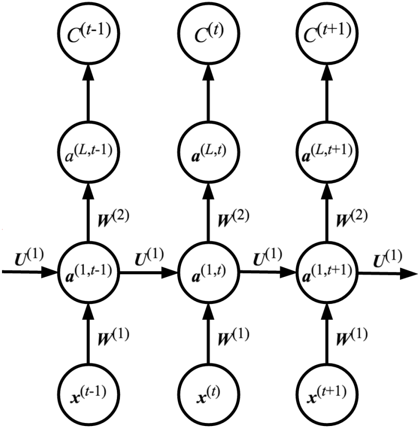
  - Bias term omitted
- $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \cdots, \mathbf{x}^{(1)}$
  - Earlier points are less important
- $\mathbf{a}^{(\cdot,t)}$'s at deeper layers give more abstract summarizations



Sequence Modeling II

$$\mathbf{a}^{(k,t)} = \mathrm{act}(\mathbf{z}^{(k,t)}) = \mathrm{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

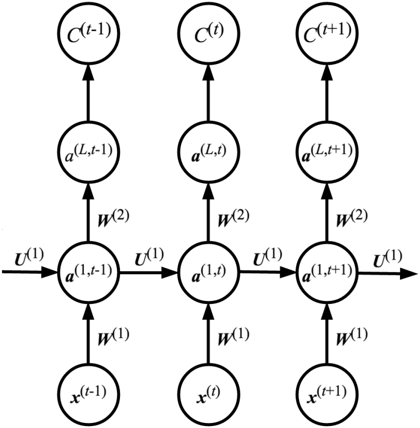
- Weights are ***shared*** across time instances



Sequence Modeling II

$$\mathbf{a}^{(k,t)} = \mathrm{act}(\mathbf{z}^{(k,t)}) = \mathrm{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

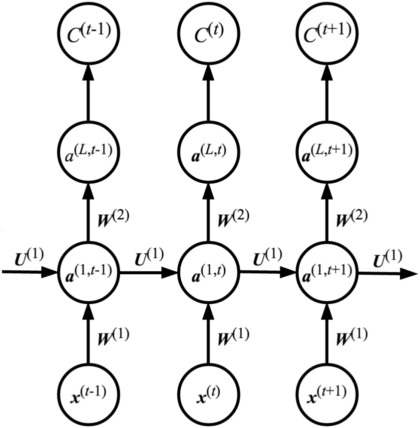
- Weights are ***shared*** across time instances
- Assumes that the "transition functions" are time invariant



Sequence Modeling II

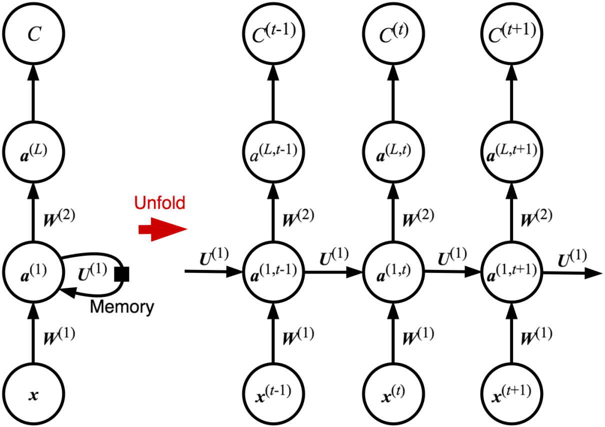
$$\mathbf{a}^{(k,t)} = \mathrm{act}(\mathbf{z}^{(k,t)})$$
$$= \mathrm{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

- Weights are ***shared*** across time instances
- Assumes that the "transition functions" are time invariant
- Our goal is to learn $\boldsymbol{U}^{(k)}$'s and $\boldsymbol{W}^{(k)}$'s for $k = 1, \cdots, L$



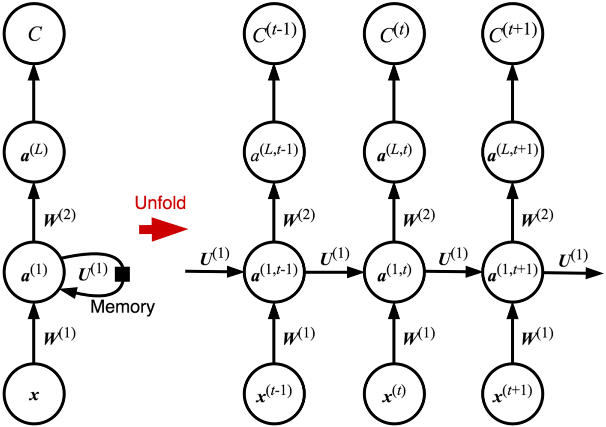
RNNs have Memory

- The computational graph of an RNN can be ***folded*** in time



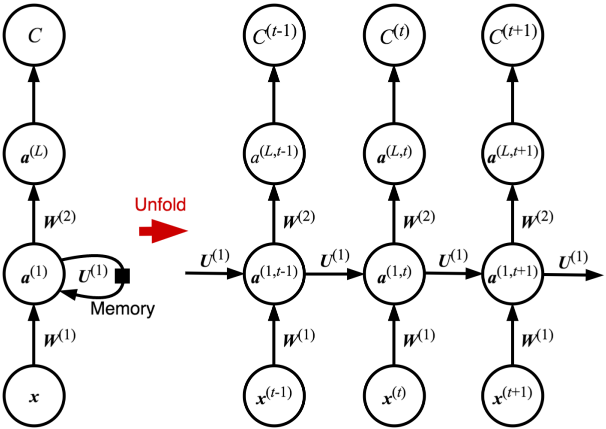
RNNs have Memory

- The computational graph of an RNN can be ***folded*** in time
- Black squares denotes ***memory*** access



Output Layer (1/2)

- With multi-class $\mathbf{y}^{(t)}$,
  - $\mathbf{a}^{(L,t)}$ represents the probability of each class
  - $C^{(t)}$ is cross entropy
- How to obtain $\hat{\mathbf{y}}^{(t)}$ rom $\mathbf{a}^{(L,t)}$ at inference time?



Output Layer (2/2)

- ***Output sampling*** for multi-class tasks:
  - Greedy: sample $\hat{\mathbf{y}}^{(t)}$ from $\mathbf{a}^{(L,t)}$
  - Bean search: sample $\hat{\mathbf{y}}^{(t)}$ from the most probable paths of the join distribution $(\mathbf{a}^{(L,t)}, \mathbf{a}^{(L,t-1)}, \cdots \mathbf{a}^{(L,t-b)})$, where b is bean size
  - Noisy: sample $\hat{\mathbf{y}}^{(t)}$ based on $\mathbf{a}^{(L,t)}$ + noise

Output Layer (2/2)

- ***Output sampling*** for multi-class tasks:
  - Greedy: sample $\hat{\mathbf{y}}^{(t)}$ from $\mathbf{a}^{(L,t)}$
  - Bean search: sample $\hat{\mathbf{y}}^{(t)}$ from the most probable paths of the join distribution $(\mathbf{a}^{(L,t)}, \mathbf{a}^{(L,t-1)}, \cdots \mathbf{a}^{(L,t-b)})$, where b is bean size
  - Noisy: sample $\hat{\mathbf{y}}^{(t)}$ based on $\mathbf{a}^{(L,t)}$ + noise
- Problem: out of vocabulary or high dimensional $\mathbf{a}^{(L,t)}$

- Solution?

Output Layer (2/2)

- ***Output sampling*** for multi-class tasks:
  - Greedy: sample $\hat{\mathbf{y}}^{(t)}$ from $\mathbf{a}^{(L,t)}$
  - Bean search: sample $\hat{\mathbf{y}}^{(t)}$ from the most probable paths of the join distribution $(\mathbf{a}^{(L,t)}, \mathbf{a}^{(L,t-1)}, \cdots \mathbf{a}^{(L,t-b)})$, where b is bean size
  - Noisy: sample $\hat{\mathbf{y}}^{(t)}$ based on $\mathbf{a}^{(L,t)}$ + noise
- Problem: out of vocabulary or high dimensional $\mathbf{a}^{(L,t)}$

- Solution? ***Subword tokenization*** (to be discussed later)

RNNs vs CNNs for Sequential Data

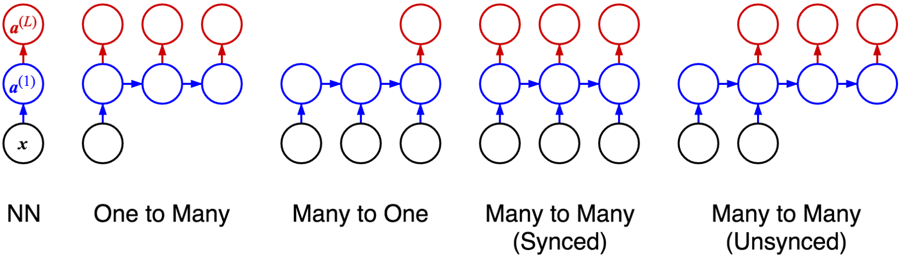
- On processing a sequence of length T at each layer with
  - D-dimensional point input and output
  - F = the CNN filter/kernel size
  - #CNN filters = D

	#Weights	Computation	Autoregressive	Point Distance
CNN	$O(FD^2)$	$O(TFD^2)$	No	$O(\frac{T}{F})$
RNN	$O(D^2)$	$O(TD^2)$	Yes	$O(T)$

Outline

Input and Output

- $\boldsymbol{x}^{(t)}$'s and $\boldsymbol{y}^{(t)}$'s do ***not*** need to have one-to-one correspondence:

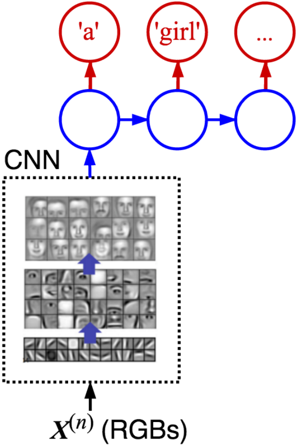


One2Many: Image Captioning

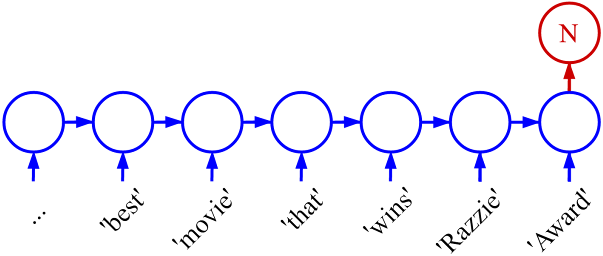
"A little girl sitting on a bed with a teddy bear."

One2Many: Image Captioning

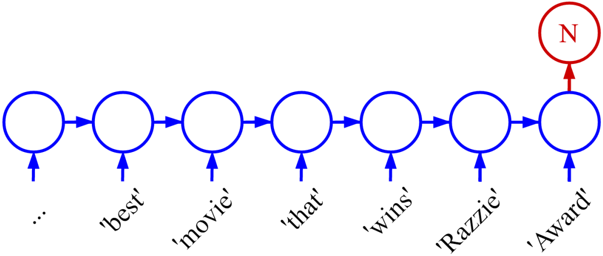
"A little girl sitting on a bed with a teddy bear."



Many2One: Sentiment Analysis



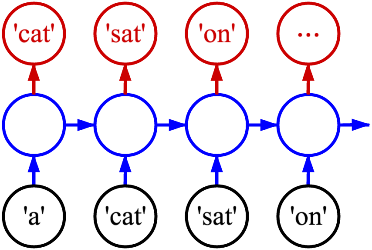
Many2One: Sentiment Analysis



- A single word (e.g., "Razzie") can negate the entire input sentence

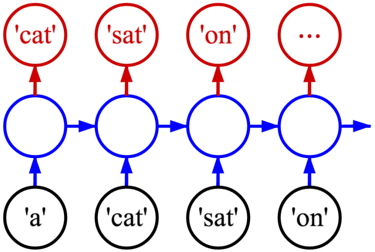
Many2Many (Synced): Language Modeling

- ***Language modeling***: predicting the next/nearby word based on the context



Many2Many (Synced): Language Modeling

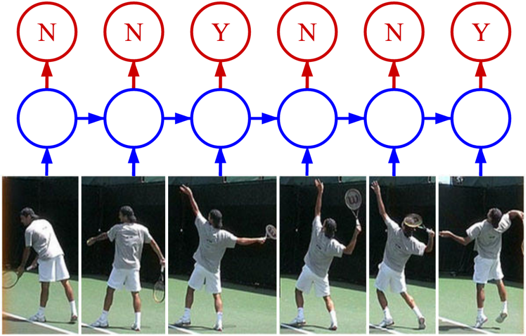
- ***Language modeling***: predicting the next/nearby word based on the context



- Latent representations of RNN provide the context

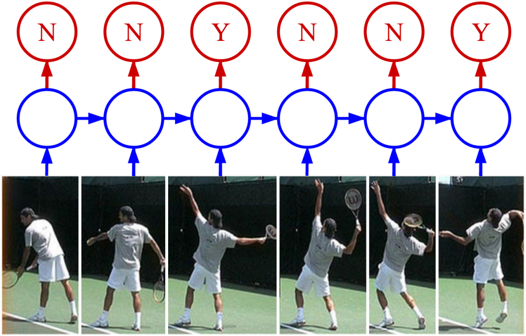
Many2Many (Synced): Video Keyframe Tagging

- Video frame annotation:



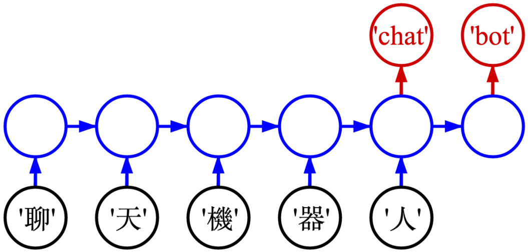
Many2Many (Synced): Video Keyframe Tagging

- Video frame annotation:



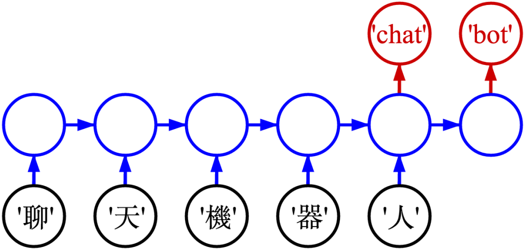
- Latent representations summarize "what's going on"

Many2Many (Unsynced): Machine Translation



- Latent representations support ***encoding*** first, and then ***decoding***
- RNN learns the structure difference

Many2Many (Unsynced): Machine Translation



- Latent representations support ***encoding*** first, and then ***decoding***
- RNN learns the structure difference
- Also called ***sequence to sequence*** learning
  - Also used in other applications, e.g., chat bots

Bidirectional RNNs

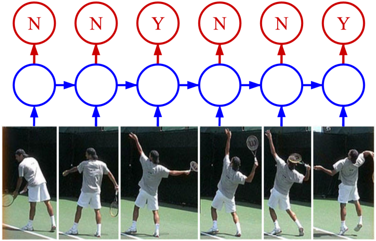
$$\mathbf{a}^{(k,t)} = \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

- Each $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \cdots, \mathbf{x}^{(1)}$

Bidirectional RNNs

$$\mathbf{a}^{(k,t)} = \mathrm{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

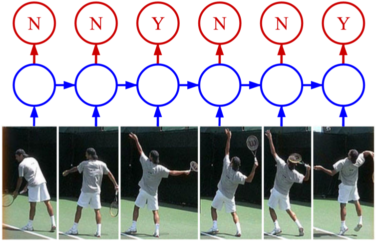
- Each $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \cdots, \mathbf{x}^{(1)}$
- A prediction is made by considering previous frames



Bidirectional RNNs

$$\mathbf{a}^{(k,t)} = \mathrm{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

- Each $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \cdots, \mathbf{x}^{(1)}$
- A prediction is made by considering previous frames
- Can we take ***future*** frames into account?

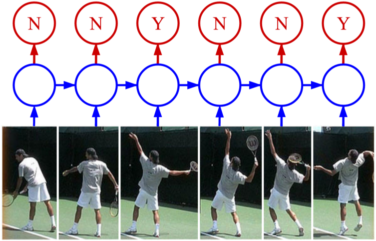


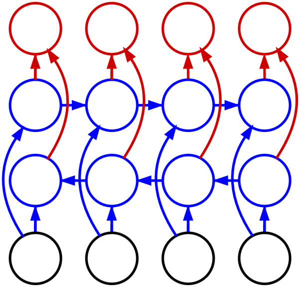
Bidirectional RNNs

$$\mathbf{a}^{(k,t)} = \text{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

- Each $\mathbf{a}^{(\cdot,t)}$ summarizes $\mathbf{x}^{(t)}, \cdots, \mathbf{x}^{(1)}$
- A prediction is made by considering previous frames
- Can we take ***future*** frames into account?
- ***Bidirectional RNNs***: output $\mathbf{a}^{(L,t)}$ depends on both $\mathbf{a}^{(k,t)}$'s and $\tilde{\mathbf{a}}^{(k,t)}$'s

$$\tilde{\mathbf{a}}^{(k,t)} = \text{act}(\tilde{\boldsymbol{U}}^{(k)}\tilde{\mathbf{a}}^{(k,t+1)} + \tilde{\boldsymbol{W}}^{(k)}\tilde{\mathbf{a}}^{(k-1,t)})$$





Recursive RNNs I

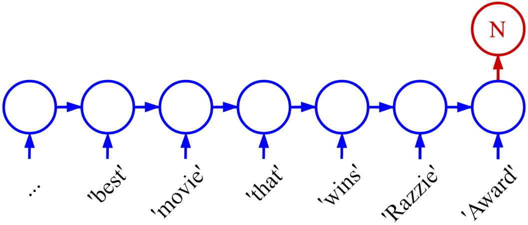
$$\mathbf{a}^{(k,t)} = \mathrm{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

- The transition of hidden representation is invariant in time
  - Earlier input/representation is less important to current $\mathbf{a}^{(\cdot,t)}$

Recursive RNNs I

$$\mathbf{a}^{(k,t)} = \mathrm{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

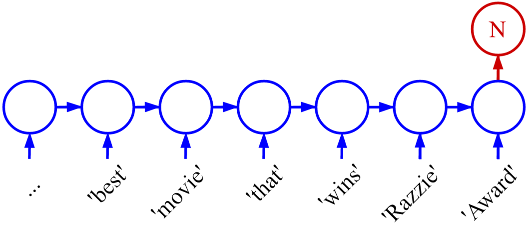
- The transition of hidden representation is invariant in time
  - Earlier input/representation is less important to current $\mathbf{a}^{(\cdot,t)}$
- "Razzie" has less effect if it is far away from the prediction



Recursive RNNs I

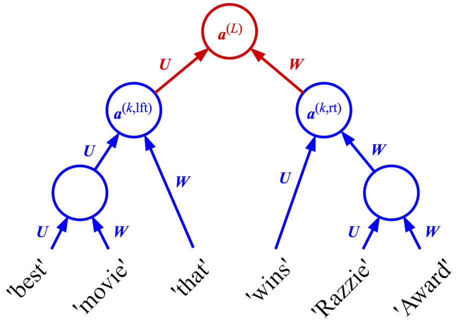
$$\mathbf{a}^{(k,t)} = \mathrm{act}(\boldsymbol{U}^{(k)}\mathbf{a}^{(k,t-1)} + \boldsymbol{W}^{(k)}\mathbf{a}^{(k-1,t)})$$

- The transition of hidden representation is invariant in time
  - Earlier input/representation is less important to current $\mathbf{a}^{(\cdot,t)}$
- "Razzie" has less effect if it is far away from the prediction
- In some applications, transitions are invariant in terms of other concepts



Recursive RNNs II

- In natural language processing (NLP), we can parse the input sentence $\boldsymbol{X}^{(n)}$ into a tree
  - Following gramma rules

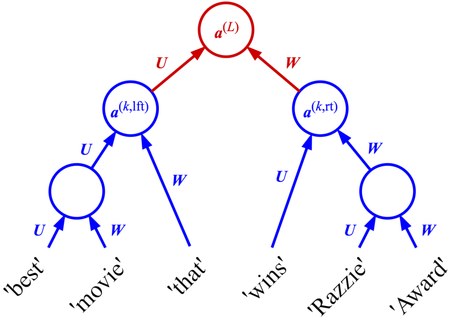


Recursive RNNs II

- In natural language processing (NLP), we can parse the input sentence $\boldsymbol{X}^{(n)}$ into a tree
  - Following gramma rules
- ***Recursive RNNs***: "subtree merges" are invariant

$$\mathbf{a}^{(k,t)} = \mathrm{act}(\boldsymbol{U}\mathbf{a}^{(k-1,\text{ left })} + \boldsymbol{W}\mathbf{a}^{(k-1,\text{ right })})$$

  - $\boldsymbol{U}$ and $\boldsymbol{W}$ are shared recursively in subtrees

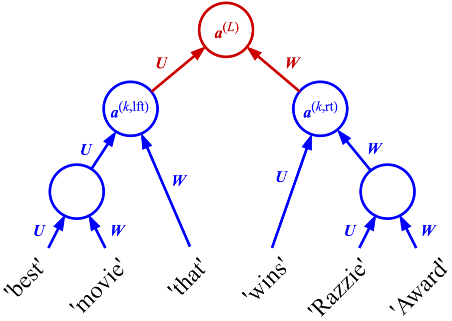


Recursive RNNs II

- In natural language processing (NLP), we can parse the input sentence $\boldsymbol{X}^{(n)}$ into a tree
  - Following gramma rules
- ***Recursive RNNs***: "subtree merges" are invariant

$$\mathbf{a}^{(k,t)} = \mathrm{act}(\boldsymbol{U}\mathbf{a}^{(k-1,\text{ left })} + \boldsymbol{W}\mathbf{a}^{(k-1,\text{ right })})$$

  - $\boldsymbol{U}$ and $\boldsymbol{W}$ are shared recursively in subtrees
- Given sentence length T, $\mathbf{a}^{(L)}$ and $\mathbf{a}^{(1,\cdot)}$ can be $O(\log T)$ away in the best case



Outline

Cost Function of Vanilla RNNs

- Parameters to learn: $\Theta = \{\boldsymbol{W}^{(k)}, \boldsymbol{U}^{(k)}\}_k$ (bias terms omitted)
- Maximum likelihood:

$$
\begin{aligned}
&\arg\min_{\Theta} C(\Theta) \\
&\quad = \arg\min_{\Theta} -\log \mathrm{P}(\mathbf{X} \,|\, \Theta) \\
&\quad = \arg\min_{\Theta} -\textstyle\sum_{n,t} \log \mathrm{P}(\boldsymbol{y}^{(n,t)} \,|\, \boldsymbol{x}^{(n,t)}, \cdots, \boldsymbol{x}^{(n,1)}, \Theta) \\
&\quad = \arg\min_{\Theta} -\textstyle\sum_{n,t} C^{(n,t)}(\Theta)
\end{aligned}
$$

  - $\mathbf{y}^{(t)}$ depends only on $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t)}$

Cost Function of Vanilla RNNs

- Parameters to learn: $\Theta = \{\boldsymbol{W}^{(k)}, \boldsymbol{U}^{(k)}\}_k$ (bias terms omitted)
- Maximum likelihood:

$$\begin{aligned}
&\arg\min_\Theta C(\Theta) \\
&\quad = \arg\min_\Theta -\log \mathrm{P}(\mathbf{X} \,|\, \Theta) \\
&\quad = \arg\min_\Theta -\textstyle\sum_{n,t} \log \mathrm{P}(\boldsymbol{y}^{(n,t)} \,|\, \boldsymbol{x}^{(n,t)}, \cdots, \boldsymbol{x}^{(n,1)}, \Theta) \\
&\quad = \arg\min_\Theta -\textstyle\sum_{n,t} C^{(n,t)}(\Theta)
\end{aligned}$$

  - $\mathbf{y}^{(t)}$ depends only on $\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(t)}$
- For example, in binary classification:
- Assuming $\mathrm{P}(\mathbf{y}^{(n,t)} = 1 \,|\, \boldsymbol{x}^{(n,t)}, \cdots, \boldsymbol{x}^{(n,1)}) \sim \mathrm{Bernoulli}(\rho^{(t)})$, we have

$$C^{(n,t)}(\Theta) = (\boldsymbol{a}^{(L,t)})^{y^{(n,t)}} (1 - \boldsymbol{a}^{(L,t)})^{(1-y^{(n,t)})}$$

  - $\boldsymbol{a}^{(L,t)} = \rho^{(t)}$ are based on $\boldsymbol{a}^{(\cdot,t)}$'s, which summarize $\boldsymbol{x}^{(n,t)}, \cdots, \boldsymbol{x}^{(n,1)}$

Outline

SGD-based Training

- RNN optimization problem can be solved using SGD:

$$\Theta^{(s+1)} \leftarrow \Theta^{(s)} - \eta \nabla_{\Theta} \sum_{n,t} C^{(n,t)}(\Theta^{(s)})$$

- Let $c^{(n,t)} = C^{(n,t)}(\Theta^{(s)})$, our goal is to evaluate $\frac{\partial c^{(n,t)}}{\partial U_{i,j}^{(k)}}$ and $\frac{\partial c^{(n,t)}}{\partial W_{i,j}^{(k)}}$
- Evaluation of $\frac{\partial c^{(n,t)}}{\partial W_{i,j}^{(k)}}$ is similar to that in DNNs and omitted
- We focus on:

$$\frac{\partial c^{(n,t)}}{\partial U_{i,j}^{(k)}} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}} \cdot \frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}} = \delta_j^{(k,t)} \frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}}$$

Forward Pass through Time

- The second term: $\frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}}$

Forward Pass through Time

- The second term: $\frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}}$
- We have $z_j^{(k,t)} = \sum_i W_{i,j}^{(k)} a_i^{(k-1,t)} + \sum_i U_{i,j}^{(k)} a_i^{(k,t-1)}$ and

$$\frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}} = a_i^{(k,t-1)}$$

Forward Pass through Time

- The second term: $\frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}}$
- We have $z_j^{(k,t)} = \sum_i W_{i,j}^{(k)} a_i^{(k-1,t)} + \sum_i U_{i,j}^{(k)} a_i^{(k,t-1)}$ and

$$\frac{\partial z_j^{(k,t)}}{\partial U_{i,j}^{(k)}} = a_i^{(k,t-1)}$$

- We can get all second terms starting from the most shallow layer and ***earliest time***

Backward Pass through Time I

- The first term (error signal): $\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}}$
- We have

$$\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \frac{\partial a_j^{(k,t)}}{\partial z_j^{(k,t)}}$$

Backward Pass through Time I

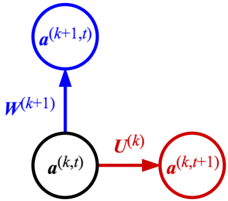
- The first term (error signal): $\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}}$
- We have

$$\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \frac{\partial a_j^{(k,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \mathrm{act}'(z_j^{(k,t)})$$

Backward Pass through Time I

- The first term (error signal): $\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}}$
- We have

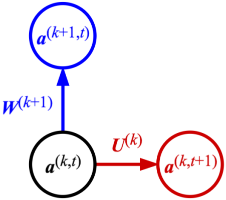
$$\begin{aligned}\delta_j^{(k,t)} &= \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \frac{\partial a_j^{(k,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \mathrm{act}'(z_j^{(k,t)}) \\ &= \left(\sum_s \frac{\partial c^{(n,t)}}{\partial z_s^{(k+1,t)}} \cdot \frac{\partial z_s^{(k+1,t)}}{\partial a_j^{(k,t)}} + \sum_s \frac{\partial c^{(n,t)}}{\partial z_s^{(k,t+1)}} \cdot \frac{\partial z_s^{(k,t+1)}}{\partial a_j^{(k,t)}} \right) \mathrm{act}'(z_j^{(k,t)})\end{aligned}$$



Backward Pass through Time I

- The first term (error signal): $\delta_j^{(k,t)} = \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}}$
- We have

$$\begin{aligned}
\delta_j^{(k,t)} &= \frac{\partial c^{(n,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \frac{\partial a_j^{(k,t)}}{\partial z_j^{(k,t)}} = \frac{\partial c^{(n,t)}}{\partial a_j^{(k,t)}} \cdot \text{act}'(z_j^{(k,t)}) \\
&= \left(\sum_s \frac{\partial c^{(n,t)}}{\partial z_s^{(k+1,t)}} \cdot \frac{\partial z_s^{(k+1,t)}}{\partial a_j^{(k,t)}} + \sum_s \frac{\partial c^{(n,t)}}{\partial z_s^{(k,t+1)}} \cdot \frac{\partial z_s^{(k,t+1)}}{\partial a_j^{(k,t)}} \right) \text{act}'(z_j^{(k,t)}) \\
&= \left(\sum_s \delta_s^{(k+1,t)} W_{j,s}^{(k+1)} + \sum_s \delta_s^{(k,t+1)} U_{j,s}^{(k)} \right) \text{act}'(z_j^{(k,t)})
\end{aligned}$$



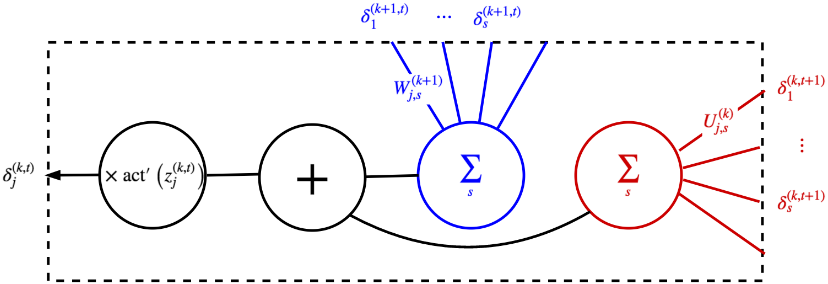
Backward Pass through Time II

$$\delta_j^{(k,t)} = \left(\sum_s \delta_s^{(k+1,t)} W_{j,s}^{(k+1)} + \sum_s \delta_s^{(k,t+1)} U_{j,s}^{(k)} \right) \mathrm{act}'(z_j^{(k,t)})$$

Backward Pass through Time II

$$\delta_j^{(k,t)} = \left(\sum_s \delta_s^{(k+1,t)} W_{j,s}^{(k+1)} + \sum_s \delta_s^{(k,t+1)} U_{j,s}^{(k)} \right) \mathrm{act}'(z_j^{(k,t)})$$

- We can evaluate all $\delta_j^{(k,t)}$'s starting from the deepest layer and ***latest time***



Backprop through Time (BPTT)

- So far, we have discussed how to compute the gradients of $U_{i,j}^{(k)}$'s (and $W_{i,j}^{(k)}$'s) for a single loss $c^{(n,t)} = C^{(n,t)}(\Theta^{(s)})$ at specific time

Backprop through Time (BPTT)

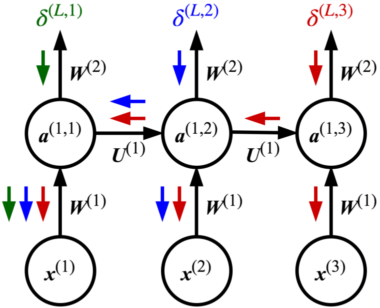
- So far, we have discussed how to compute the gradients of $U_{i,j}^{(k)}$'s (and $W_{i,j}^{(k)}$'s) for a single loss $c^{(n,t)} = C^{(n,t)}(\Theta^{(s)})$ at specific time
- However, for each sequence n, we have multiple $c^{(n,t)}$'s at different t's
  - Their gradient need to be summed

Backprop through Time (BPTT)

- So far, we have discussed how to compute the gradients of $U_{i,j}^{(k)}$'s (and $W_{i,j}^{(k)}$'s) for a single loss $c^{(n,t)} = C^{(n,t)}(\Theta^{(s)})$ at specific time
- However, for each sequence n, we have multiple $c^{(n,t)}$'s at different t's
  - Their gradient need to be summed
- For different $c^{(n,\cdot)}$'s, the forward pass can be shared

Backprop through Time (BPTT)

- So far, we have discussed how to compute the gradients of $U_{i,j}^{(k)}$'s (and $W_{i,j}^{(k)}$'s) for a single loss $c^{(n,t)} = C^{(n,t)}(\Theta^{(s)})$ at specific time
- However, for each sequence n, we have multiple $c^{(n,t)}$'s at different t's
  - Their gradient need to be summed
- For different $c^{(n,\cdot)}$'s, the forward pass can be shared
- ***BPTT***: single forward pass, ***multiple*** backward passes



Outline

Long-Term Dependencies

- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(\boldsymbol{z}^{(k,t)})$$

- Backward pass:

$$\boldsymbol{\delta}^{(k,t)} \leftarrow \mathrm{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\boldsymbol{\delta}^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\boldsymbol{\delta}^{(k+1,t)})$$

Long-Term Dependencies

- Forward pass:

$$z^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(z^{(k,t)})$$

- Backward pass:

$$\delta^{(k,t)} \leftarrow \mathrm{act}'(z^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\delta^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\delta^{(k+1,t)})$$

- The dependency between $\boldsymbol{a}^{(k,i)}$ (resp. $\delta^{(k,i)}$) and $\boldsymbol{a}^{(k,j)}$ (resp. $\delta^{(k,j)}$) are maintained by $\left(\boldsymbol{U}^{(k)}\right)^{(j-i)}$
    - Ignoring activation function and depth, we have $\boldsymbol{a}^{(k,j)} = (\boldsymbol{U}^{(k)\top})^{j-i}\boldsymbol{a}^{(k,i)}$ and $\delta^{(k,i)} = (\boldsymbol{U}^{(k)})^{j-i}\delta^{(k,j)}$

Long-Term Dependencies

- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(\boldsymbol{z}^{(k,t)})$$

- Backward pass:

$$\delta^{(k,t)} \leftarrow \mathrm{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\delta^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\delta^{(k+1,t)})$$

- The dependency between $\boldsymbol{a}^{(k,i)}$ (resp. $\delta^{(k,i)}$) and $\boldsymbol{a}^{(k,j)}$ (resp. $\delta^{(k,j)}$) are maintained by $\left(\boldsymbol{U}^{(k)}\right)^{(j-i)}$
  - Ignoring activation function and depth, we have $\boldsymbol{a}^{(k,j)} = (\boldsymbol{U}^{(k)\top})^{j-i}\boldsymbol{a}^{(k,i)}$ and $\delta^{(k,i)} = (\boldsymbol{U}^{(k)})^{j-i}\delta^{(k,j)}$
- If $\boldsymbol{a}^{(k,i)}$ and $\boldsymbol{a}^{(k,j)}$ are far away in time, their long-term dependency causes optimization problems

Exploding/Vanishing Gradient Problem

- Ignoring activation function and depth:

$$\boldsymbol{a}^{(k,j)} = (\boldsymbol{U}^{(k)\top})^{(j-i)}\boldsymbol{a}^{(k,i)} \text{ and } \boldsymbol{\delta}^{(k,i)} = (\boldsymbol{U}^{(k)})^{(j-i)}\boldsymbol{\delta}^{(k,j)}$$

- Given eigendecomposition: $\boldsymbol{U}^{(k)} = \boldsymbol{Q}\begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}} \end{bmatrix}\boldsymbol{Q}^\top$

Exploding/Vanishing Gradient Problem

- Ignoring activation function and depth:

$$\boldsymbol{a}^{(k,j)} = (\boldsymbol{U}^{(k)\top})^{(j-i)}\boldsymbol{a}^{(k,i)} \text{ and } \boldsymbol{\delta}^{(k,i)} = (\boldsymbol{U}^{(k)})^{(j-i)}\boldsymbol{\delta}^{(k,j)}$$

- Given eigendecomposition: $\boldsymbol{U}^{(k)} = \boldsymbol{Q}\begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}} \end{bmatrix}\boldsymbol{Q}^\top$
- We have: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q}\begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix}\boldsymbol{Q}^\top$

Exploding/Vanishing Gradient Problem

- Ignoring activation function and depth:

$$\boldsymbol{a}^{(k,j)} = (\boldsymbol{U}^{(k)\top})^{(j-i)}\boldsymbol{a}^{(k,i)} \text{ and } \delta^{(k,i)} = (\boldsymbol{U}^{(k)})^{(j-i)}\delta^{(k,j)}$$

- Given eigendecomposition: $\boldsymbol{U}^{(k)} = \boldsymbol{Q}\begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}} \end{bmatrix}\boldsymbol{Q}^\top$
- We have: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q}\begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix}\boldsymbol{Q}^\top$
- When $j-i$ is large, λ_s^{j-i} is either very large or small
  - $\lambda_s = 1.01 \Rightarrow \lambda_s^{1000} \approx 20,000$
  - $\lambda_s = 0.99 \Rightarrow \lambda_s^{1000} \approx 0$

Exploding/Vanishing Gradient Problem

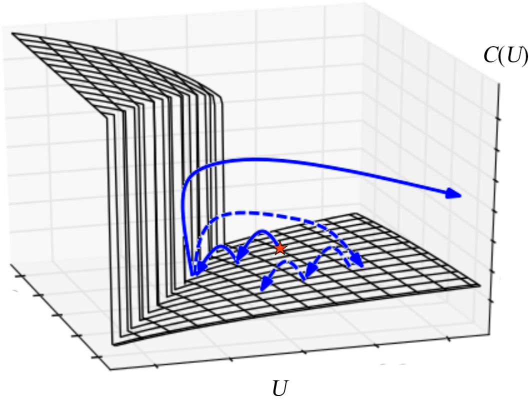
- Ignoring activation function and depth:

$$\boldsymbol{a}^{(k,j)} = (\boldsymbol{U}^{(k)\top})^{(j-i)}\boldsymbol{a}^{(k,i)} \text{ and } \delta^{(k,i)} = (\boldsymbol{U}^{(k)})^{(j-i)}\delta^{(k,j)}$$

- Given eigendecomposition: $\boldsymbol{U}^{(k)} = \boldsymbol{Q}\begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}} \end{bmatrix}\boldsymbol{Q}^\top$
- We have: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q}\begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix}\boldsymbol{Q}^\top$
- When $j-i$ is large, λ_s^{j-i} is either very large or small
  - $\lambda_s = 1.01 \Rightarrow \lambda_s^{1000} \approx 20,000$
  - $\lambda_s = 0.99 \Rightarrow \lambda_s^{1000} \approx 0$
- Exploding or vanishing gradients!

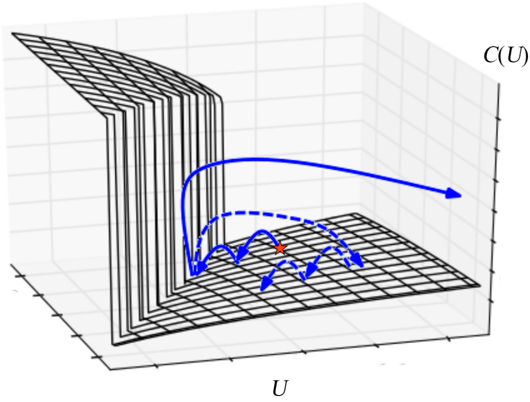
Cost Surface

- The cost surface of C is either very flat or steep
- Hard for gradient-based optimization



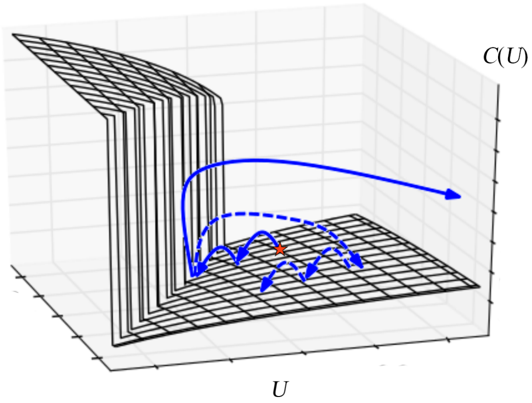
Cost Surface

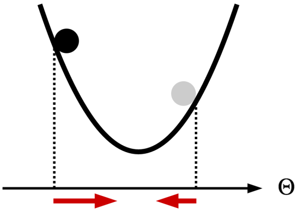
- The cost surface of C is either very flat or steep
- Hard for gradient-based optimization
- Optimization techniques?



Nesterov Momentum

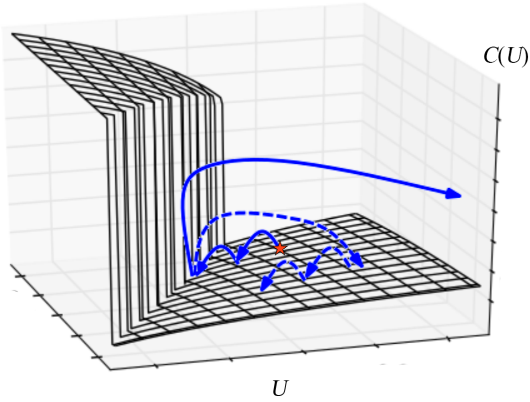
- Use Nesterov momentum to “brake” before hitting the wall





Gradient Clipping

- A simple way is to avoid the exploding gradient problem is to ***clip*** a gradient if it exceeds a predefined threshold
- Very effective in practice

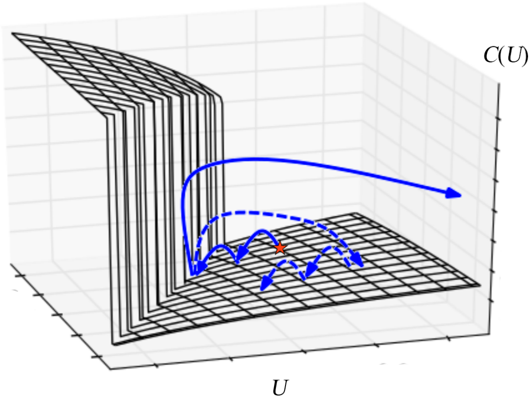


RMS Prop

- Adaptive learning rate based on statistics of recent gradients:

$$\boldsymbol{r}^{(t+1)} \leftarrow \lambda \boldsymbol{r}^{(t)} + (1-\lambda)\boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$$

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{r}^{(t+1)}}} \odot \boldsymbol{g}^{(t)}$$



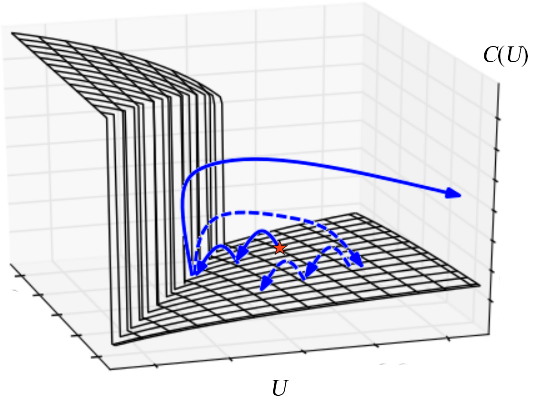
RMS Prop

- Adaptive learning rate based on statistics of recent gradients:

$$\boldsymbol{r}^{(t+1)} \leftarrow \lambda \boldsymbol{r}^{(t)} + (1-\lambda)\boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$$

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{r}^{(t+1)}}} \odot \boldsymbol{g}^{(t)}$$

- Reduce λ



Outline

Sigmoid Activation Function

- Traditionally, the hidden units of RNNs use the sigmoid activation functions
  - E.g., $\tanh(\cdot)$ the ***hyperbolic tangent***

Sigmoid Activation Function

- Traditionally, the hidden units of RNNs use the sigmoid activation functions
  - E.g., $\tanh(\cdot)$ the ***hyperbolic tangent***
- Why sigmoid activation function?

Sigmoid Activation Function

- Traditionally, the hidden units of RNNs use the sigmoid activation functions
  - E.g., $\tanh(\cdot)$ the ***hyperbolic tangent***
- Why sigmoid activation function?
- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(\boldsymbol{z}^{(k,t)})$$

Backward pass:

$$\boldsymbol{\delta}^{(k,t)} \leftarrow \mathrm{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\boldsymbol{\delta}^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\boldsymbol{\delta}^{(k+1,t)})$$

Sigmoid Activation Function

- Traditionally, the hidden units of RNNs use the sigmoid activation functions
  - E.g., $\tanh(\cdot)$ the ***hyperbolic tangent***
- Why sigmoid activation function?
- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(\boldsymbol{z}^{(k,t)})$$

  Backward pass:

$$\boldsymbol{\delta}^{(k,t)} \leftarrow \mathrm{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\boldsymbol{\delta}^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\boldsymbol{\delta}^{(k+1,t)})$$

- Sigmoid activation:
  - Bounded range in the forward pass
  - $\mathrm{act}'(\cdot) < 1$ in the backward pass

Sigmoid Activation Function

- Traditionally, the hidden units of RNNs use the sigmoid activation functions
  - E.g., $\tanh(\cdot)$ the ***hyperbolic tangent***
- Why sigmoid activation function?
- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(\boldsymbol{z}^{(k,t)})$$

  Backward pass:

$$\boldsymbol{\delta}^{(k,t)} \leftarrow \mathrm{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\boldsymbol{\delta}^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\boldsymbol{\delta}^{(k+1,t)})$$

- Sigmoid activation:
  - Bounded range in the forward pass
  - $\mathrm{act}'(\cdot) < 1$ in the backward pass
- Mitigates the exploding (but not vanishing) gradient problem for $\boldsymbol{U}^{(k)}$'s

Sigmoid Activation Function

- Traditionally, the hidden units of RNNs use the sigmoid activation functions
  - E.g., $\tanh(\cdot)$ the ***hyperbolic tangent***
- Why sigmoid activation function?
- Forward pass:

$$\boldsymbol{z}^{(k,t)} \leftarrow \boldsymbol{U}^{(k)\top}\boldsymbol{a}^{(k,t-1)} + \boldsymbol{W}^{(k)\top}\boldsymbol{a}^{(k-1,t)}$$

$$\boldsymbol{a}^{(k,t)} \leftarrow \mathrm{act}(\boldsymbol{z}^{(k,t)})$$

  Backward pass:

$$\boldsymbol{\delta}^{(k,t)} \leftarrow \mathrm{act}'(\boldsymbol{z}^{(k,t)}) \odot (\boldsymbol{U}^{(k)}\boldsymbol{\delta}^{(k,t+1)} + \boldsymbol{W}^{(k+1)}\boldsymbol{\delta}^{(k+1,t)})$$

- Sigmoid activation:
  - Bounded range in the forward pass
  - $\mathrm{act}'(\cdot) < 1$ in the backward pass
- Mitigates the exploding (but not vanishing) gradient problem for $\boldsymbol{U}^{(k)}$'s
- Introduces vanishing gradients of $\boldsymbol{W}^{(k)}$'s

Learning Unitary $\boldsymbol{U}^{(k)}$'s

- Long-term dependency: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q} \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \boldsymbol{Q}^\top$

Learning Unitary $U^{(k)}$'s

- Long-term dependency: $(U^{(k)})^{j-i} = Q \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} Q^\top$
- Why not make $U^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?

Learning Unitary $U^{(k)}$'s

- Long-term dependency: $(U^{(k)})^{j-i} = Q \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} Q^\top$
- Why not make $U^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?
- Hinton et al. [10] propose IRNN:
  - ***Initializes $U^{(k)} = I$*** in SGD
  - Uses ***ReLU*** hidden units

Learning Unitary $U^{(k)}$'s

- Long-term dependency: $(U^{(k)})^{j-i} = Q \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} Q^\top$
- Why not make $U^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?
- Hinton et al. [10] propose IRNN:
  - ***Initializes $U^{(k)} = I$*** in SGD
  - Uses ***ReLU*** hidden units
  - Empirically, requires a very small learning rate (e.g., 10^{-8}) to work well
  - Simple, but very slow

Learning Unitary $U^{(k)}$'s

- Long-term dependency: $(\boldsymbol{U}^{(k)})^{j-i} = \boldsymbol{Q} \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} \boldsymbol{Q}^\top$
- Why not make $\boldsymbol{U}^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?
- Hinton et al. [10] propose IRNN:
  - ***Initializes $\boldsymbol{U}^{(k)} = \boldsymbol{I}$*** in SGD
  - Uses ***ReLU*** hidden units
  - Empirically, requires a very small learning rate (e.g., 10^{-8}) to work well
  - Simple, but very slow
- Krueger et al. [7] add a term $\sum_{k,t}(\|\boldsymbol{a}^{(k,t)}\| - \|\boldsymbol{a}^{(k,t-1)}\|)^2$ to IRNN cost to stabilize the norms of $\boldsymbol{a}^{(k,t)}$'s in time

Learning Unitary $U^{(k)}$'s

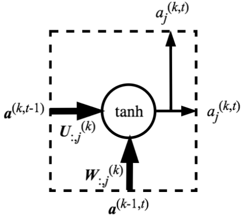
- Long-term dependency: $(U^{(k)})^{j-i} = Q \begin{bmatrix} \lambda_1^{j-i} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{D^{(k)}}^{j-i} \end{bmatrix} Q^\top$
- Why not make $U^{(k)}$'s unitary (i.e., $\lambda_s = 1$ for all s)?
- Hinton et al. [10] propose IRNN:
  - ***Initializes $U^{(k)} = I$*** in SGD
  - Uses ***ReLU*** hidden units
  - Empirically, requires a very small learning rate (e.g., 10^{-8}) to work well
  - Simple, but very slow
- Krueger et al. [7] add a term $\sum_{k,t}(\|a^{(k,t)}\| - \|a^{(k,t-1)}\|)^2$ to IRNN cost to stabilize the norms of $a^{(k,t)}$'s in time
- Bengio et al. [1] propose uRNN that learns unitary $U^{(k)}$'s explicitly

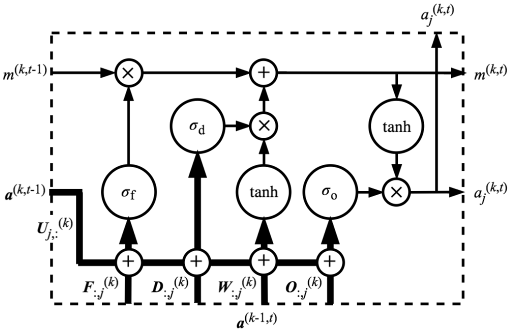
Long Short-Term Memory (LSTM)

- Idea: to create ***shortcut*** in each neuron for the error signals to flow backward more smoothly

Long Short-Term Memory (LSTM)

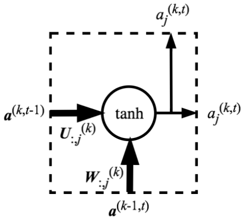
- Idea: to create ***shortcut*** in each neuron for the error signals to flow backward more smoothly
- The j-th LSTM unit at depth k and time t:

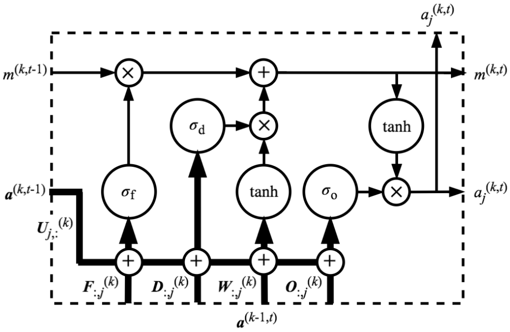




Long Short-Term Memory (LSTM)

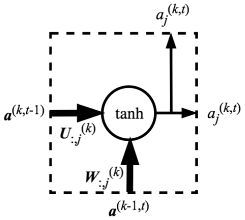
- Idea: to create ***shortcut*** in each neuron for the error signals to flow backward more smoothly
- The j-th LSTM unit at depth k and time t:
  - First tanh activation to be added into the ***memory cell*** $m^{(k,t)}$
  - Second tanh activation as the final activation $a_j^{(k,t)}$

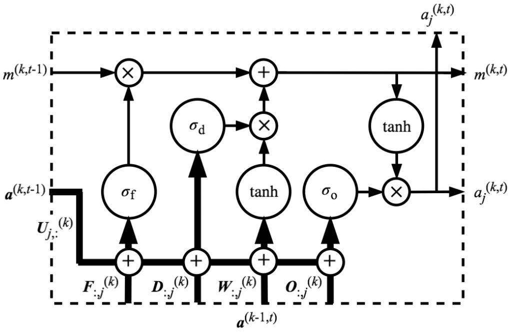




Long Short-Term Memory (LSTM)

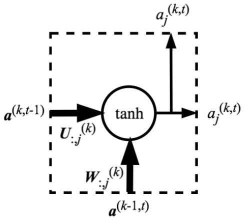
- Idea: to create ***shortcut*** in each neuron for the error signals to flow backward more smoothly
- The j-th LSTM unit at depth k and time t:
  - First tanh activation to be added into the ***memory cell*** $m^{(k,t)}$
  - Second tanh activation as the final activation $a_j^{(k,t)}$
  - Forget gate σ_f: whether to forget $m^{(k,t-1)}$

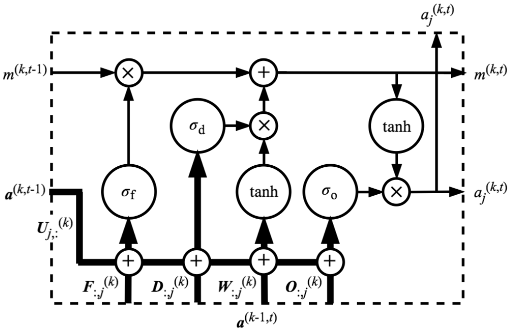




Long Short-Term Memory (LSTM)

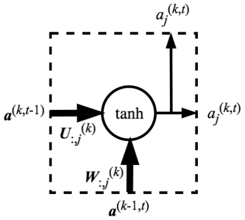
- Idea: to create ***shortcut*** in each neuron for the error signals to flow backward more smoothly
- The j-th LSTM unit at depth k and time t:
  - First tanh activation to be added into the ***memory cell*** $m^{(k,t)}$
  - Second tanh activation as the final activation $a_j^{(k,t)}$
  - Forget gate σ_f: whether to forget $m^{(k,t-1)}$
  - Input gate σ_d: whether to store the first activation into $m^{(k,t)}$

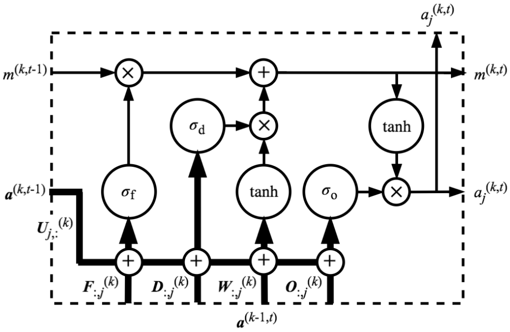




Long Short-Term Memory (LSTM)

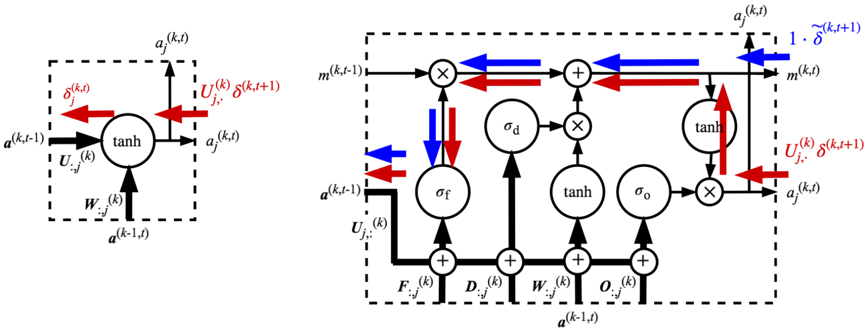
- Idea: to create ***shortcut*** in each neuron for the error signals to flow backward more smoothly
- The j-th LSTM unit at depth k and time t:
  - First tanh activation to be added into the ***memory cell*** $m^{(k,t)}$
  - Second tanh activation as the final activation $a_j^{(k,t)}$
  - Forget gate σ_{f}: whether to forget $m^{(k,t-1)}$
  - Input gate σ_{d}: whether to store the first activation into $m^{(k,t)}$
  - Output gate σ_{o}: whether to output the second activation





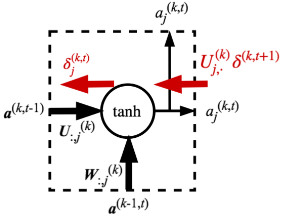
Error Signals

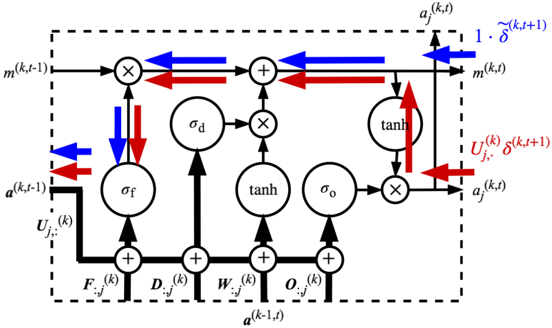
- Error signals now have a second path
  - If the forget gate is open, error signals won't decay (***blue arrows***)



Error Signals

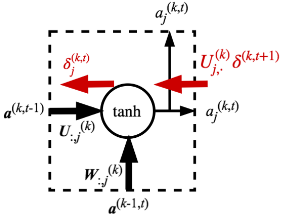
- Error signals now have a second path
  - If the forget gate is open, error signals won't decay (***blue arrows***)
  - Avoids the vanishing gradients (but not exploding ones)

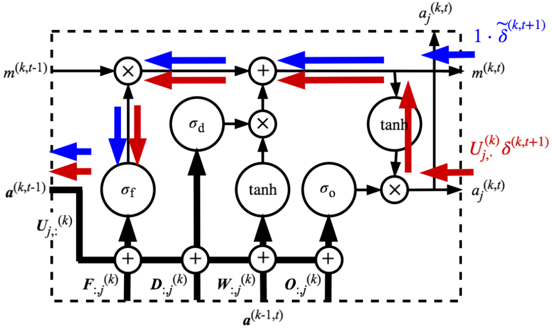




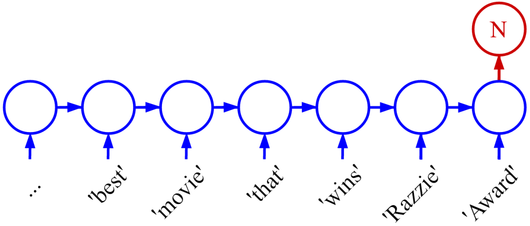
Error Signals

- Error signals now have a second path
  - If the forget gate is open, error signals won't decay (***blue arrows***)
  - Avoids the vanishing gradients (but not exploding ones)
- When NN decides to close the forget gate, the vanishing gradient problem is irrelevant
- In practice, LSTM + gradient clipping works well together



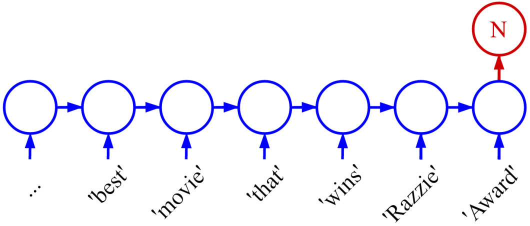


Dynamic Representations for Sentiment Analysis



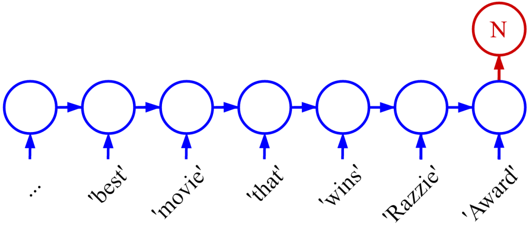
- LSTM units learn dynamic representations

Dynamic Representations for Sentiment Analysis



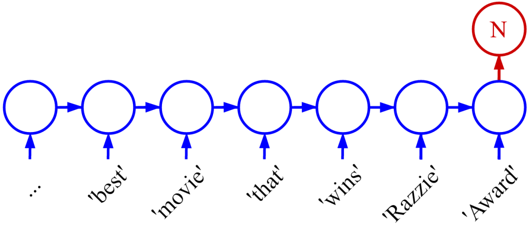
- LSTM units learn dynamic representations
- Closing forget gate for "Razzie"
  - To correct previous summarization and/or shift focus

Dynamic Representations for Sentiment Analysis



- LSTM units learn dynamic representations
- Closing forget gate for "Razzie"
  - To correct previous summarization and/or shift focus
- Closing input gate for "movie"
  - Not to learn, to keep the same summarization

Dynamic Representations for Sentiment Analysis

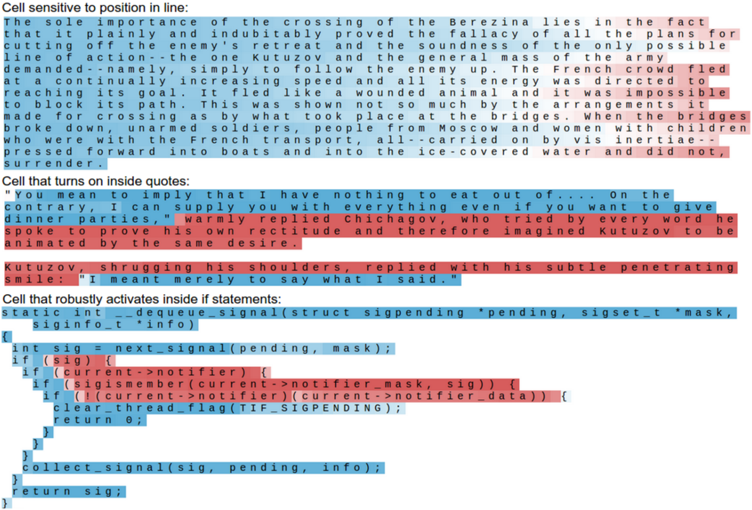


- LSTM units learn dynamic representations
- Closing forget gate for "Razzie"
  - To correct previous summarization and/or shift focus
- Closing input gate for "movie"
  - Not to learn, to keep the same summarization
- Closing output gate for "that"
  - To let the next neuron decide the activation/gate values by its own

Dynamic Representations for Language Models

Dynamic Representations for Language Models

- Neuron activations for language modeling [6] Interactive Tool



Learning Process of LSTMs

- Output at epoch 100:

  "*... tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh...*"

Learning Process of LSTMs

- Output at epoch 100:

  "... tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh..."

- At 300 (spaces and periods):

  "... Phe lism thond hon at. MeiDimorotion in ther..."

Learning Process of LSTMs

- Output at epoch 100:

  "*... tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh...*"

- At 300 (spaces and periods):

  "*... Phe lism thond hon at. MeiDimorotion in ther...*"

- At 500 (common words "we," "he," etc.):

  "*... we counter. He stutn co des. His stanted out one...*"

Learning Process of LSTMs

- Output at epoch 100:

  "*... tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh...*"

- At 300 (spaces and periods):

  "*... Phe lism thond hon at. MeiDimorotion in ther...*"
- At 500 (common words "we," "he," etc.):

  "*... we counter. He stutn co des. His stanted out one...*"
- At 700 (English-like structure):

  "*... Aftair fall unsuch that the hall for Prince Velzonski's...*"

Learning Process of LSTMs

- Output at epoch 100:

  "... tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh..."

- At 300 (spaces and periods):

  "... Phe lism thond hon at. MeiDimorotion in ther..."

- At 500 (common words "we," "he," etc.):

  "... we counter. He stutn co des. His stanted out one..."

- At 700 (English-like structure):

  "... Aftair fall unsuch that the hall for Prince Velzonski's..."

- At 1200 (quotations and longer words):

  "... "Kite vouch!" he repeated by her door..."

Learning Process of LSTMs

- Output at epoch 100:

  "*... tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh...*"

- At 300 (spaces and periods):

  "*... Phe lism thond hon at. MeiDimorotion in ther...*"
- At 500 (common words "we," "he," etc.):

  "*... we counter. He stutn co des. His stanted out one...*"
- At 700 (English-like structure):

  "*... Aftair fall unsuch that the hall for Prince Velzonski's...*"
- At 1200 (quotations and longer words):

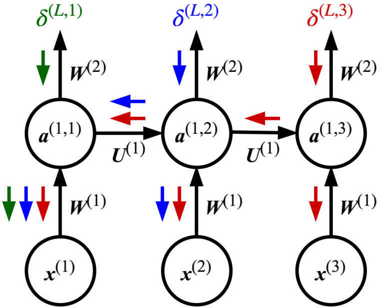
  "*... "Kite vouch!" he repeated by her door...*"
- At 2000 (topics and longer-term dependencies):

  "*... "Why do what that day," replied Natasha, ...*"

Outline

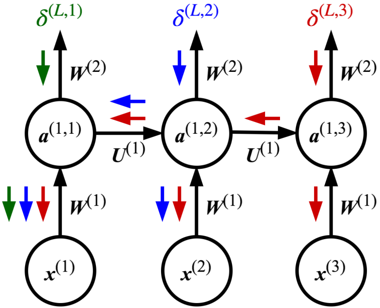
Parallelism

- A forward/backward pass through time in BPTT cannot be parallelized



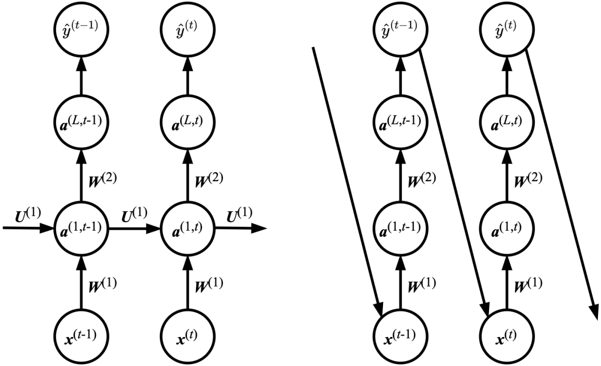
Parallelism

- A forward/backward pass through time in BPTT cannot be parallelized
- The ***hidden-to-hidden*** recurrent connections in a vanilla RNN create dependency between
  - $\boldsymbol{a}^{(k,t)}$'s in forward pass
  - $\delta^{(k,t)}$'s in backward pass



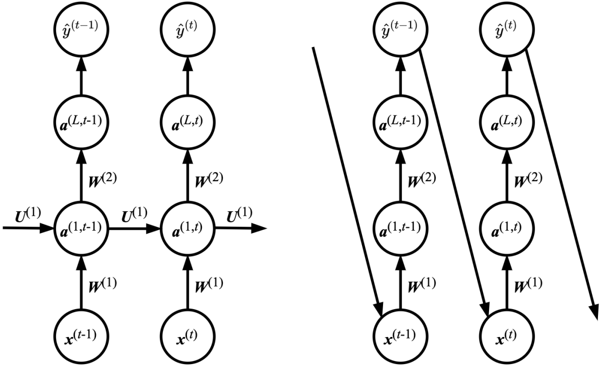
Output Recurrence and Teacher Forcing

- *Teacher forcing*: replace hidden-to-hidden recurrence with output-to-hidden or output-to-input recurrence



Output Recurrence and Teacher Forcing

- ***Teacher forcing***: replace hidden-to-hidden recurrence with output-to-hidden or output-to-input recurrence
- At training time, use ***correct labels*** $\boldsymbol{y}^{(\cdot)}$**'s** to train the model
  - So, the forward/backward pass through time can be parallelized
- At test time, switch back to using model output $\hat{\boldsymbol{y}}^{(\cdot)}$'s



Exposure Bias

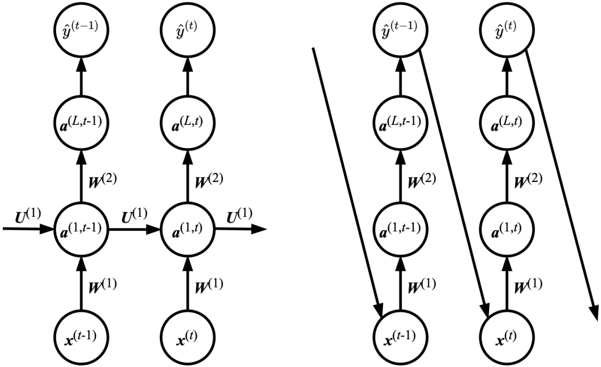
- Mismatch between $\boldsymbol{y}^{(\cdot)}$'s and $\hat{\boldsymbol{y}}^{(\cdot)}$'s hurts RNN performance
- Solution?

Exposure Bias

- Mismatch between $\boldsymbol{y}^{(\cdot)}$'s and $\hat{\boldsymbol{y}}^{(\cdot)}$'s hurts RNN performance
- Solution? Scheduled sampling
- At training time,
  1. Use $\boldsymbol{y}^{(\cdot)}$'s initially
  2. Gradually mix in $\hat{\boldsymbol{y}}^{(\cdot)}$'s later

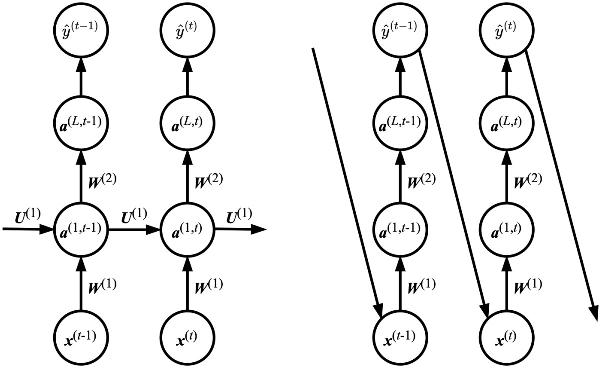
Reduced Expressiveness

- The vanilla RNNs are universal in the sense that they can simulate Turing machines [18]



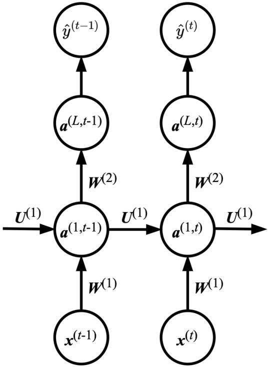
Reduced Expressiveness

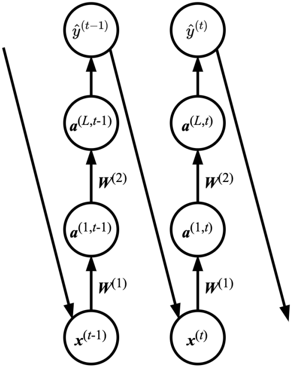
- The vanilla RNNs are universal in the sense that they can simulate Turing machines [18]
- Output-recurrent RNNs ***cannot*** simulate Turing machines and are strictly less powerful



Reduced Expressiveness

- The vanilla RNNs are universal in the sense that they can simulate Turing machines [18]
- Output-recurrent RNNs ***cannot*** simulate Turing machines and are strictly less powerful
- The output $\boldsymbol{a}^{(L,\cdot)}$'s are explicitly trained to match training targets
    - Cannot capture all required information in the past to predict the future





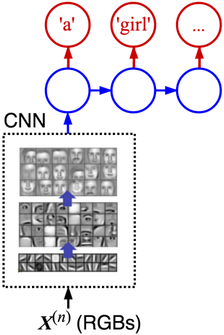
Outline

Outline

Limited Representation Size

- In some RNNs, a hidden representation $\mathbf{a}^{(\cdot,t)}$ needs to support:
  - Current prediction $\mathbf{a}^{(L,t)}$, and
  - ***All*** future predictions $\mathbf{a}^{(L,t+1)}, \cdots, \mathbf{a}^{(L,T)}$

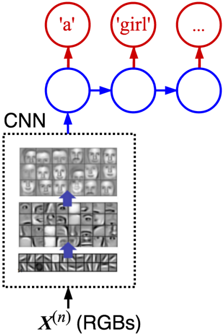
"A little girl sitting on a bed with a teddy bear."



Limited Representation Size

- In some RNNs, a hidden representation $\mathbf{a}^{(\cdot,t)}$ needs to support:
  - Current prediction $\mathbf{a}^{(L,t)}$, and
  - ***All*** future predictions $\mathbf{a}^{(L,t+1)}, \cdots, \mathbf{a}^{(L,T)}$
- The fixed-size $\mathbf{a}^{(t)}$ faces a trade-off between:
  - Representing face features for current prediction ("girl")
  - Representing other features for future predictions ("bear")

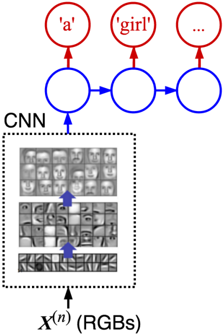
"A little girl sitting on a bed with a teddy bear."



Limited Representation Size

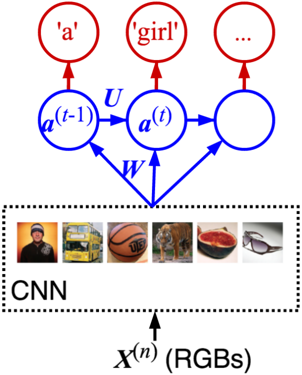
- In some RNNs, a hidden representation $\mathbf{a}^{(\cdot,t)}$ needs to support:
  - Current prediction $\mathbf{a}^{(L,t)}$, and
  - ***All*** future predictions $\mathbf{a}^{(L,t+1)}, \cdots, \mathbf{a}^{(L,T)}$
- The fixed-size $\mathbf{a}^{(t)}$ faces a trade-off between:
  - Representing face features for current prediction ("girl")
  - Representing other features for future predictions ("bear")
- Can we ease the job of $\mathbf{a}^{(\cdot,t)}$?

"A little girl sitting on a bed with a teddy bear."



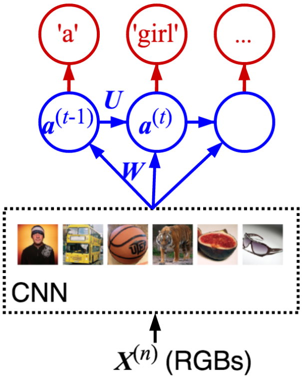
Input Recurrence

- ***Input recurrence***: to feed the entire input $\mathbf{X}$ to all $\mathbf{a}^{(1,t)}$'s, $\forall t$



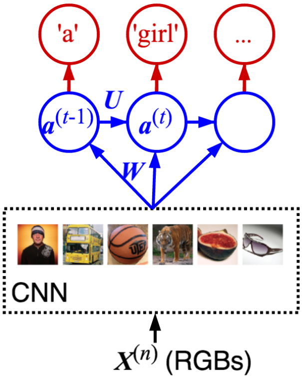
Input Recurrence

- ***Input recurrence***: to feed the entire input $\mathbf{X}$ to all $\mathbf{a}^{(1,t)}$'s, $\forall t$
- Now, $\mathbf{a}^{(\cdot,t)}$ only needs to:
  - Support current prediction $\mathbf{a}^{(L,t)}$, and
  - Provide context to the ***next*** representation $\mathbf{a}^{(\cdot,t+1)}$



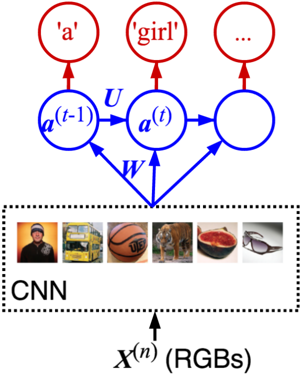
Input Recurrence

- ***Input recurrence***: to feed the entire input $\mathbf{X}$ to all $\mathbf{a}^{(1,t)}$'s, $\forall t$
- Now, $\mathbf{a}^{(\cdot,t)}$ only needs to:
  - Support current prediction $\mathbf{a}^{(L,t)}$, and
  - Provide context to the ***next*** representation $\mathbf{a}^{(\cdot,t+1)}$
- E.g., when predicting "girl," $\boldsymbol{a}^{(\cdot,t)}$ may pay attention to only few face-related images features of current input $\boldsymbol{X}^{(n)}$



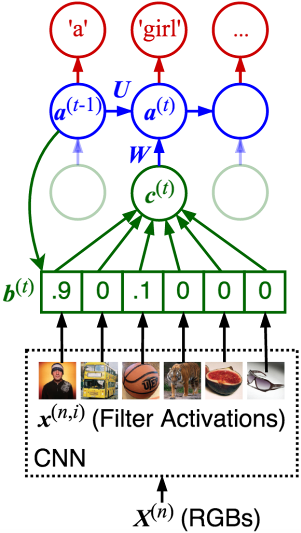
Input Recurrence

- ***Input recurrence***: to feed the entire input $\mathbf{X}$ to all $\mathbf{a}^{(1,t)}$'s, $\forall t$
- Now, $\mathbf{a}^{(\cdot,t)}$ only needs to:
  - Support current prediction $\mathbf{a}^{(L,t)}$, and
  - Provide context to the ***next*** representation $\mathbf{a}^{(\cdot,t+1)}$
- E.g., when predicting "girl," $\boldsymbol{a}^{(\cdot,t)}$ may pay attention to only few face-related images features of current input $\boldsymbol{X}^{(n)}$
- Why not model the attention explicitly?
  - So we can see where $\mathbf{a}^{(\cdot,t)}$ is "looking at"



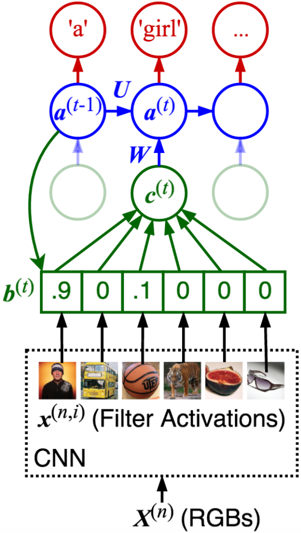
Attention Mechanism

- Assumes that the input $\mathbf{X} = \{\mathbf{x}^{(i)}\}_i$ can be broken into "parts"
  - E.g., with CNN, $\mathbf{x}^{(i)}$ could be the activation values of a filter



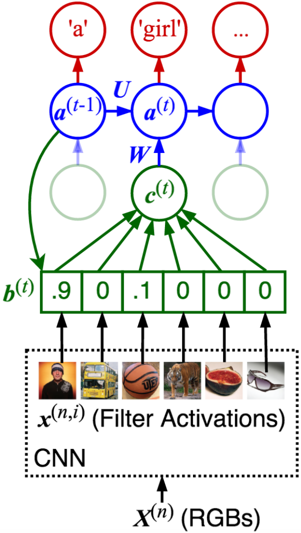
Attention Mechanism

- Assumes that the input $\mathbf{X} = \{\mathbf{x}^{(i)}\}_i$ can be broken into "parts"
  - E.g., with CNN, $\mathbf{x}^{(i)}$ could be the activation values of a filter
- For each timestamp t:
  - Obtain from $\mathbf{a}^{(L-1,t-1)}$ an ***attention vector*** $\mathbf{b}^{(t)}$, $\sum \mathrm{b}_i^{(t)} = 1$



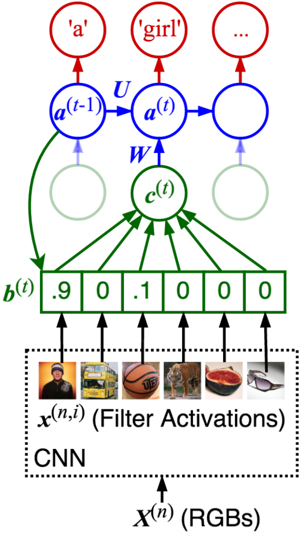
Attention Mechanism

- Assumes that the input $\mathbf{X} = \{\mathbf{x}^{(i)}\}_i$ can be broken into "parts"
  - E.g., with CNN, $\mathbf{x}^{(i)}$ could be the activation values of a filter
- For each timestamp t:
  - Obtain from $\mathbf{a}^{(L-1,t-1)}$ an ***attention vector*** $\mathbf{b}^{(t)}$, $\sum \mathrm{b}_i^{(t)} = 1$
  - Feed $\mathbf{a}^{(1,t)}$ with the weighted input $\mathbf{c}^{(t)} = \sum_i \mathrm{b}_i^{(t)} \mathbf{x}^{(i)}$



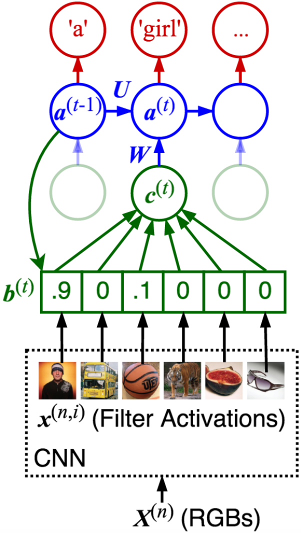
Attention Mechanism

- Assumes that the input $\mathbf{X} = \{\mathbf{x}^{(i)}\}_i$ can be broken into "parts"
  - E.g., with CNN, $\mathbf{x}^{(i)}$ could be the activation values of a filter
- For each timestamp t:
  - Obtain from $\mathbf{a}^{(L-1,t-1)}$ an ***attention vector*** $\mathbf{b}^{(t)}$, $\sum \mathrm{b}_i^{(t)} = 1$
  - Feed $\mathbf{a}^{(1,t)}$ with the weighted input $\mathbf{c}^{(t)} = \sum_i \mathrm{b}_i^{(t)} \mathbf{x}^{(i)}$
- Reduces size of $\boldsymbol{W}$ from $O(|\mathbf{X}| \cdot |\mathbf{a}^{(\cdot,t)}|)$ to $O(|\mathbf{x}^{(i)}| \cdot |\mathbf{a}^{(\cdot,t)}|)$



Attention Mechanism

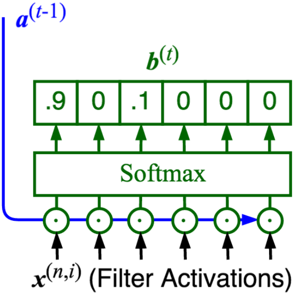
- Assumes that the input $\mathbf{X} = \{\mathbf{x}^{(i)}\}_i$ can be broken into "parts"
  - E.g., with CNN, $\mathbf{x}^{(i)}$ could be the activation values of a filter
- For each timestamp t:
  - Obtain from $\mathbf{a}^{(L-1,t-1)}$ an ***attention vector*** $\mathbf{b}^{(t)}$, $\sum \mathrm{b}_i^{(t)} = 1$
  - Feed $\mathbf{a}^{(1,t)}$ with the weighted input $\mathbf{c}^{(t)} = \sum_i \mathrm{b}_i^{(t)} \mathbf{x}^{(i)}$
- Reduces size of $\boldsymbol{W}$ from $O(|\mathbf{X}| \cdot |\mathbf{a}^{(\cdot,t)}|)$ to $O(|\mathbf{x}^{(i)}| \cdot |\mathbf{a}^{(\cdot,t)}|)$
- How to obtain $\mathbf{b}^{(t)}$?



Computing Attention Vector

1. Use $\mathbf{a}^{(L-1,t-1)}$ as a "query" to get a match score for each input part by using, e.g., a simple NN [2, 20]:

$$z_i = \mathrm{act}(\boldsymbol{p}^\top \mathbf{a}^{(L-1,t-1)} + \boldsymbol{q}^\top \mathbf{x}^{(i)} + r)$$



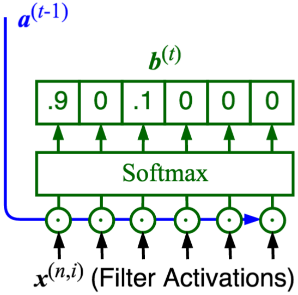
Computing Attention Vector

1. Use $\mathbf{a}^{(L-1,t-1)}$ as a "query" to get a match score for each input part by using, e.g., a simple NN [2, 20]:

$$\mathrm{z}_i = \mathrm{act}(\boldsymbol{p}^\top \mathbf{a}^{(L-1,t-1)} + \boldsymbol{q}^\top \mathbf{x}^{(i)} + r)$$

2. Normalize and concentrate on few larger scores by:

$$\mathrm{b}_i = \mathrm{softmax}(\mathrm{z})_i = \frac{\exp(\mathrm{z}_i)}{\sum_j \exp(\mathrm{z}_j)}$$



Computing Attention Vector

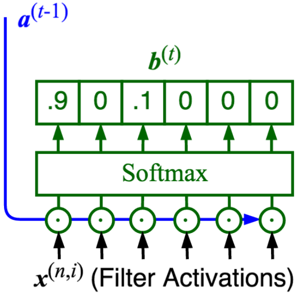
1. Use $\mathbf{a}^{(L-1,t-1)}$ as a "query" to get a match score for each input part by using, e.g., a simple NN [2, 20]:

$$z_i = \mathrm{act}(\boldsymbol{p}^\top \mathbf{a}^{(L-1,t-1)} + \boldsymbol{q}^\top \mathbf{x}^{(i)} + r)$$

   - Jointly trained with the main RNN

2. Normalize and concentrate on few larger scores by:

$$b_i = \mathrm{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$



Computing Attention Vector

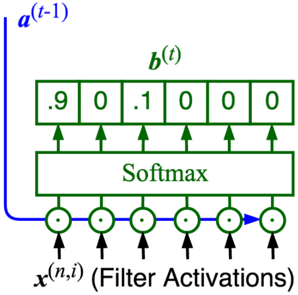
1. Use $\mathbf{a}^{(L-1,t-1)}$ as a "query" to get a match score for each input part by using, e.g., a simple NN [2, 20]:

$$z_i = \mathrm{act}(\boldsymbol{p}^\top \mathbf{a}^{(L-1,t-1)} + \boldsymbol{q}^\top \mathbf{x}^{(i)} + r)$$

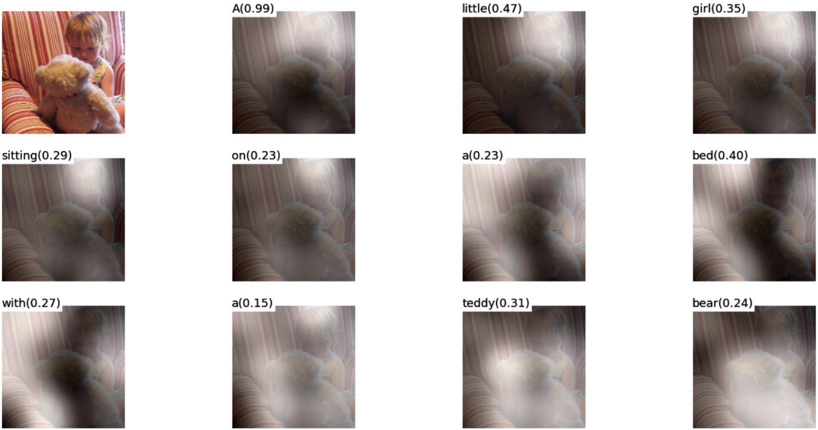
   - Jointly trained with the main RNN
   - $\boldsymbol{p}$, $\boldsymbol{q}$, and r are shared by different i's and t's (weight tying)

2. Normalize and concentrate on few larger scores by:

$$b_i = \mathrm{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

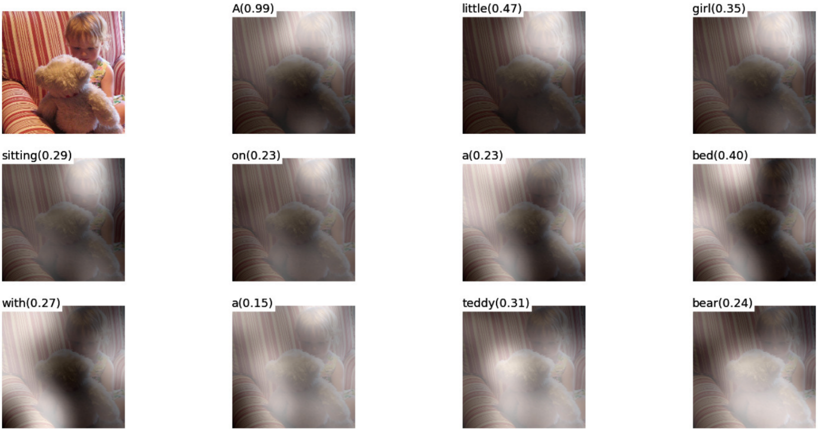


Visualizing Attention



- How to draw a mask?

Visualizing Attention

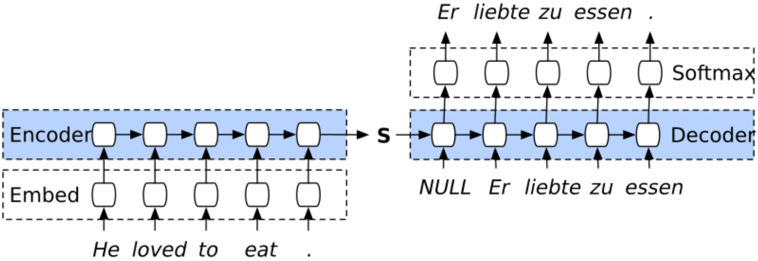


- How to draw a mask? Threat $\mathbf{c}^{(t)} = \sum_i \mathrm{b}_i^{(t)} \mathbf{x}^{(i)}$ as image and enlarge it

Outline

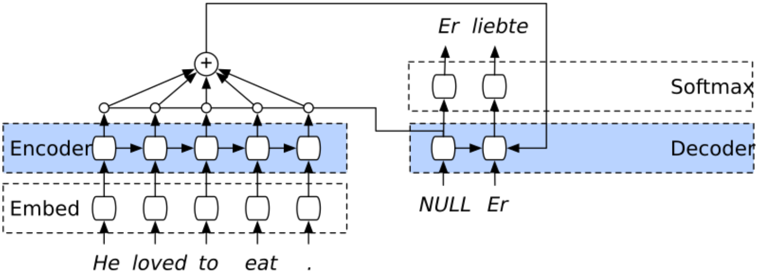
GoogleNMTv1: Encoder-Decoder

- LSTM-based RNN
- Hidden state S encodes an entire input sequence
- Then supplied to the decoder



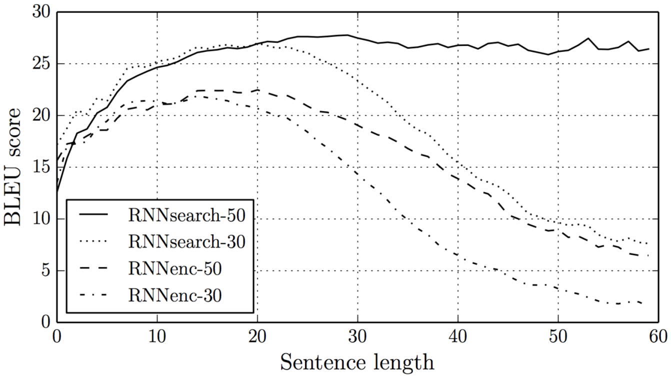
GoogleNMTv2: Attention-based Encoder-Decoder

- No feed-forward connection between the encoder and decoder
- Instead, uses previous output as query to get attention and next output
  - Allows for retrieving different parts of input sentence depending on decoding context



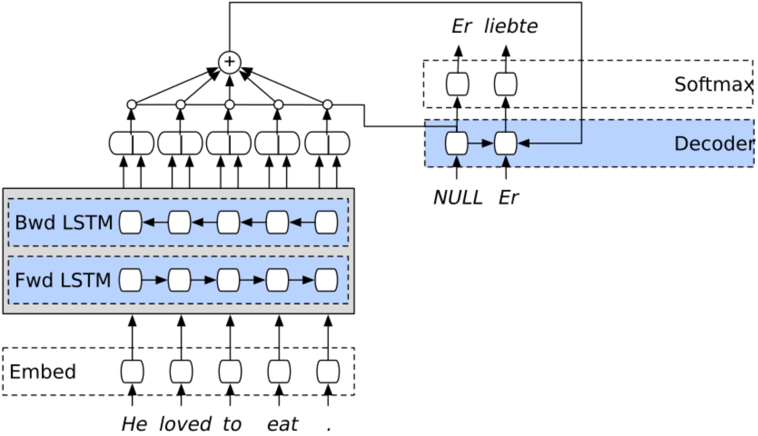
Long Sequences

- Attention-based model generates long sequences better



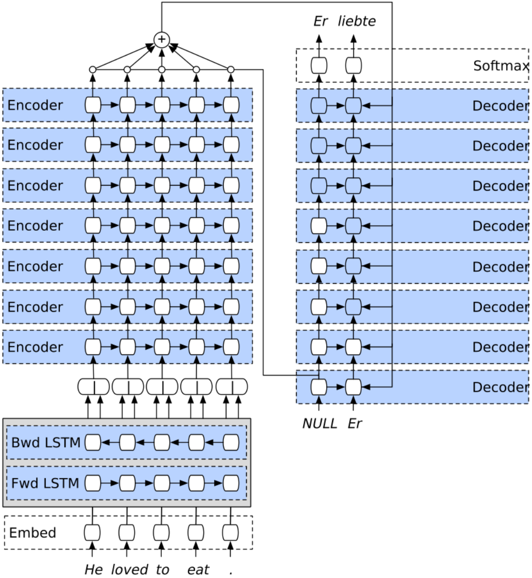
GoogleNMTv3: Bidirectional Encoder Layer

- Takes into account future words when summarizing input sequence
- Better determines the meaning/context



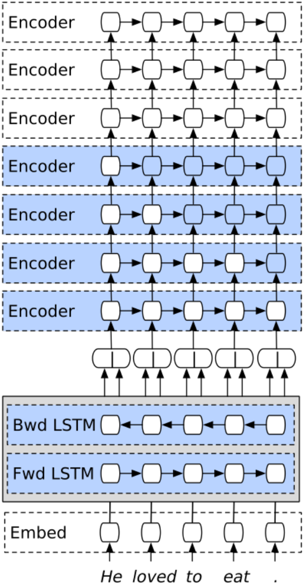
GoogleNMTv4: Going Deep

- Encoder:
  - ***1*** bi-directional layer
  - 7 uni-directional layers
- Decoder:
  - 8 uni-directional layers
  - ***Lowest*** decoder layer for querying attention

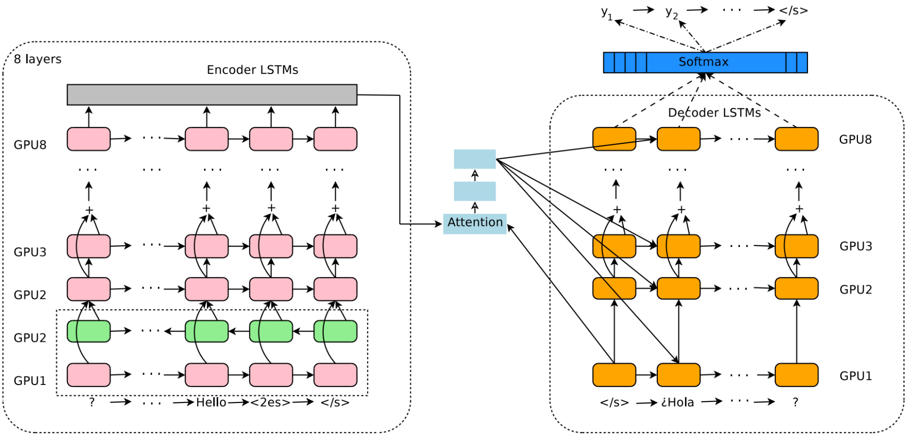


GoogleNMTv5: Parallelization

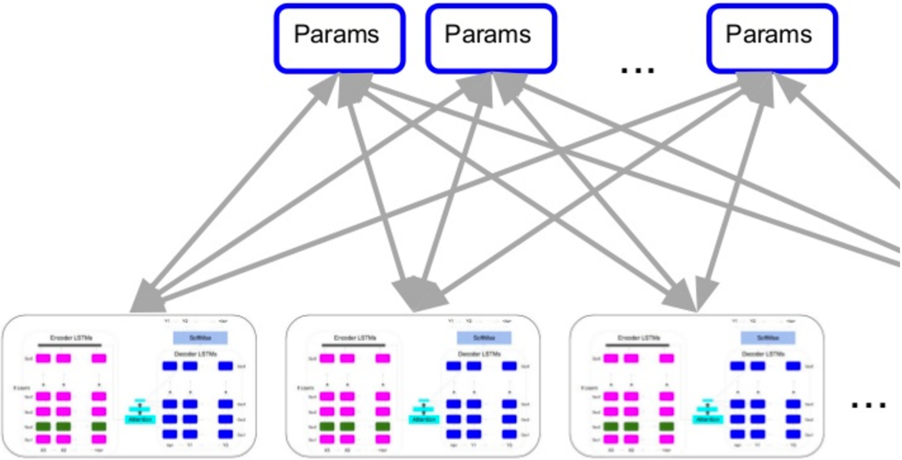
- Each layer trained by a GPU
- Forward pass:
  - Encoder: 7 uni-directional encoder layers trained in a pipeline
  - Decoder: pipeline starts as soon as encoder layers are ready
- Backward pass:
  - Teacher forcing



Model Parallelism (1 Machine, 8 GPUs)

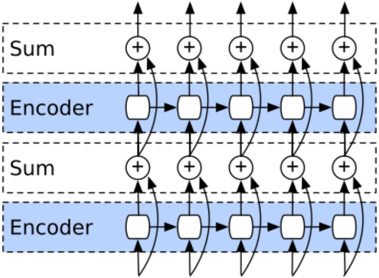


Data Parallelism (Multiple Param Servers)



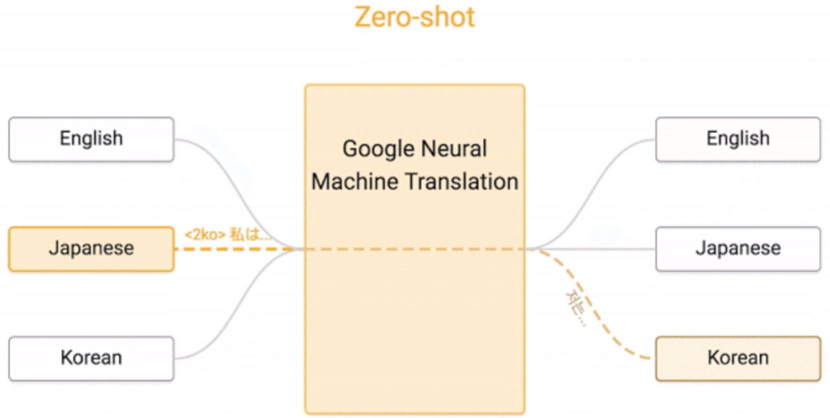
GoogleNMTv6: Residuals

- Upper layer learns the ***delta*** function to the lower one
- Easier to train a deep NN



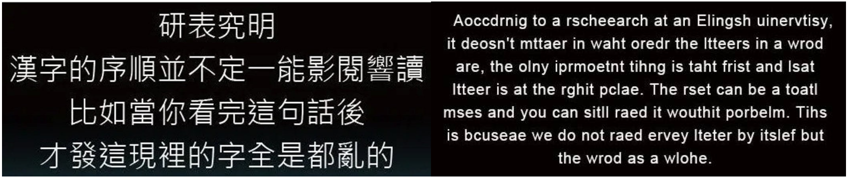
GoogleNMTv7: Multilingual & Zero-Shot Translation

- Training: input x augmented with task identifier (language pair)
  - E.g., (eng, jp), (kr, en), (jp, en)
- Inference: ***unknown*** language-pair identifier
  - E.g., (kr, jp)

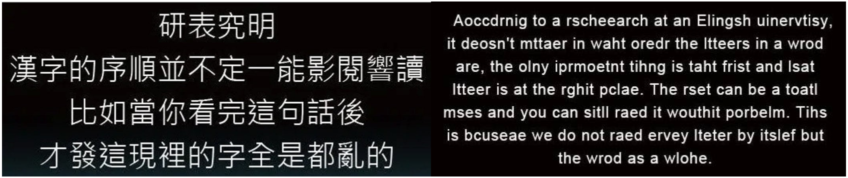


Outline

How Humans Process Text?



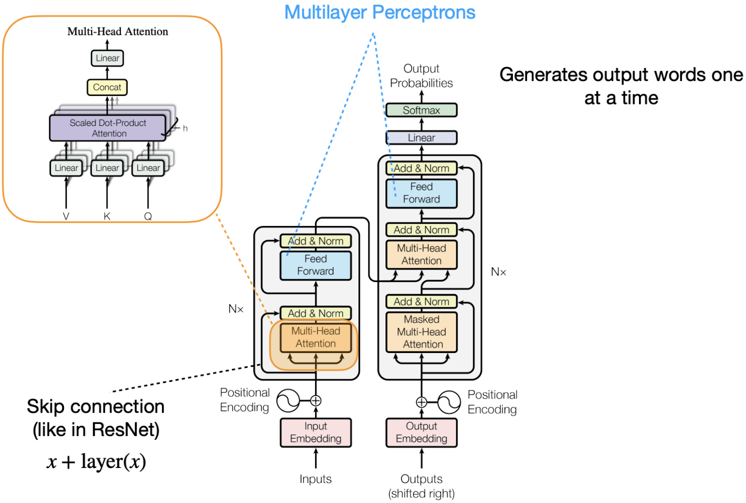
How Humans Process Text?



- ***Not entirely sequential*** as in RNNs
- ***Not entirely based on local patterns*** as in CNNs
- Any other “better” architectures?

Outline

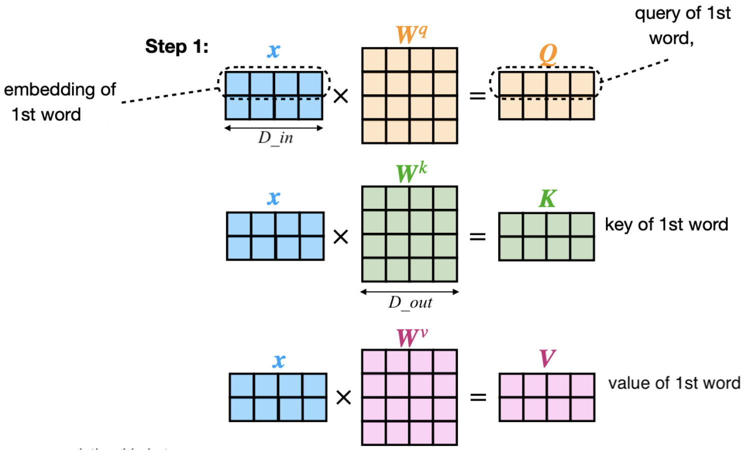
Attention is All You Need [19]



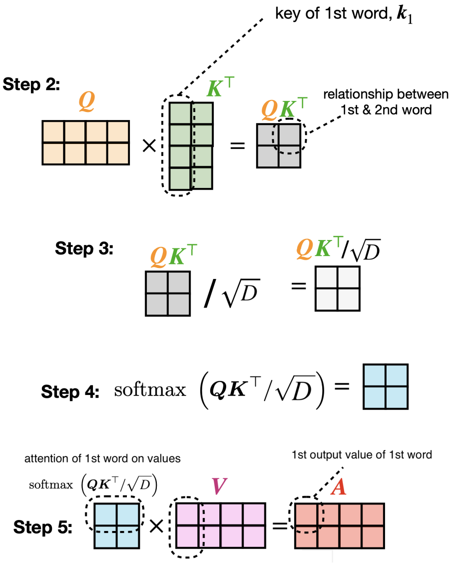
- RNN layers are replaced with ***self-*** and ***cross-attention*** blocks

Self-Attention

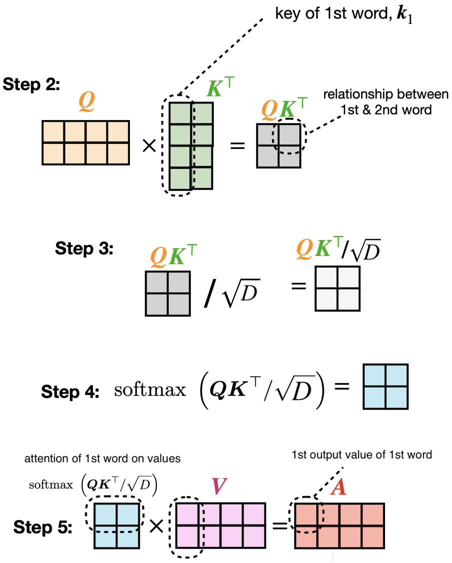
- Weights to learn at each layer: $\boldsymbol{W}_{\text{query}}, \boldsymbol{W}_{\text{key}}, \boldsymbol{W}_{\text{value}} \in \mathbb{R}^{D \times D}$
- Batch, non-autoregressive processing of sequences



Self-Attention

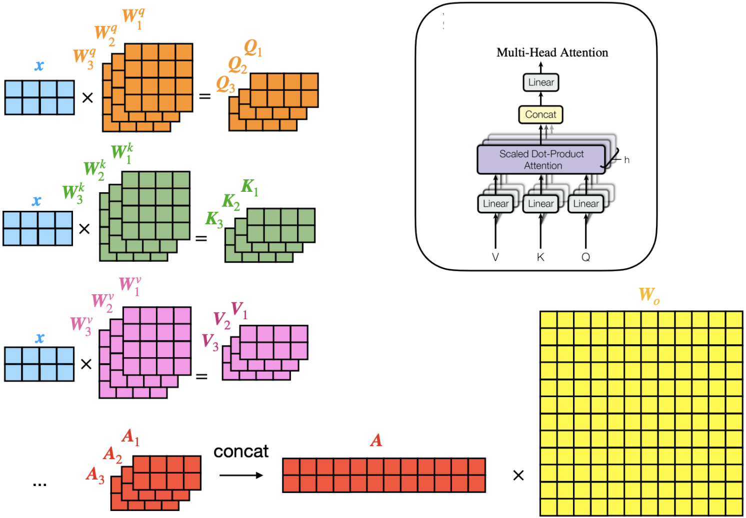


Self-Attention



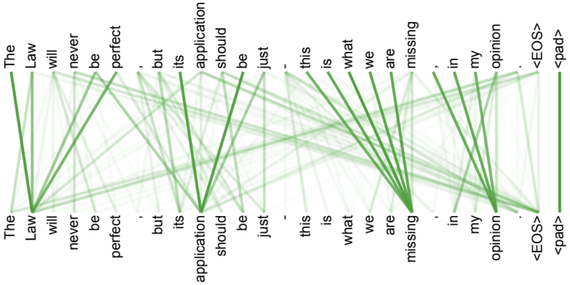
- Input: $\boldsymbol{A}^{(l-1)} \in \mathbb{R}^{T \times D}$
- Output: $\boldsymbol{A}^{(l)} \in \mathbb{R}^{T \times D}$
  - ***Context*** augmented
  - E.g, "**train** a model" vs. "get on a **train**"

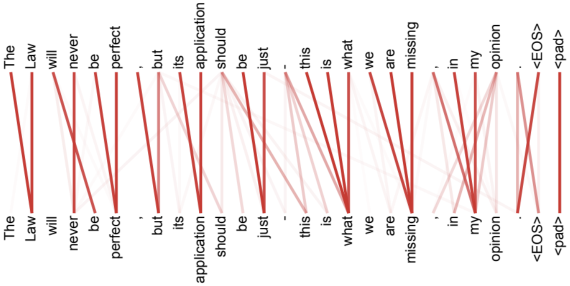
Multi-head Self-Attention



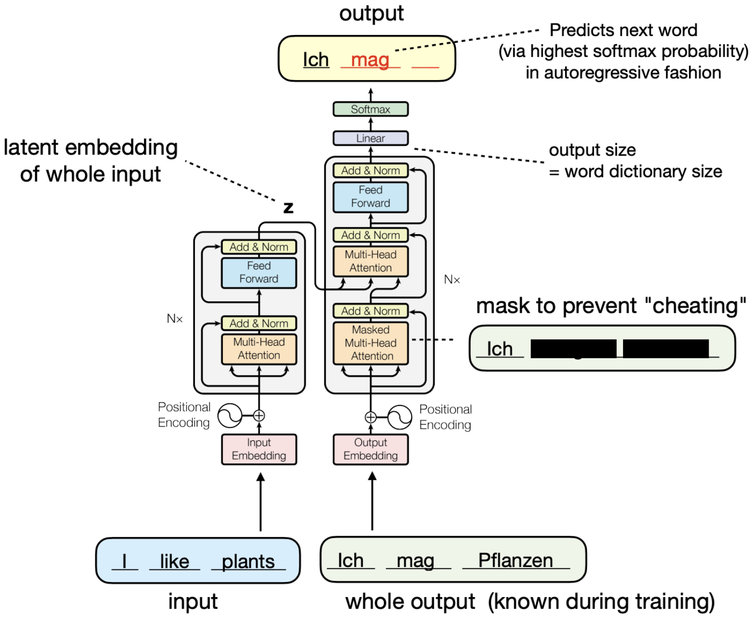
- Given H heads, we have $\boldsymbol{W}_{\text{query}}^{(h)}, \boldsymbol{W}_{\text{key}}^{(h)}, \boldsymbol{W}_{\text{value}}^{(h)} \in \mathbb{R}^{D \times \frac{D}{H}}$ and $\boldsymbol{A} \in \mathbb{R}^{T \times D}$

Same Sequence, Different Context Augmentations



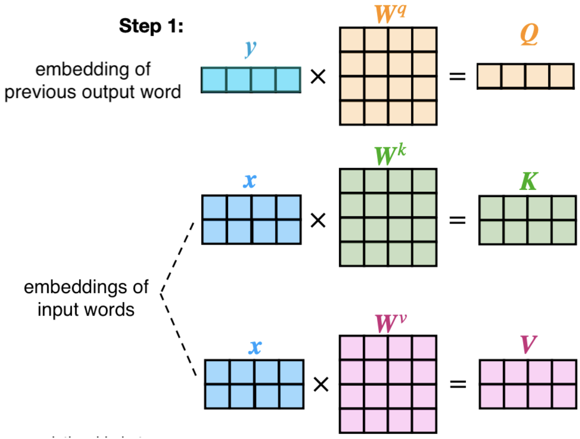


Masked Self-Attention for Decoding Output

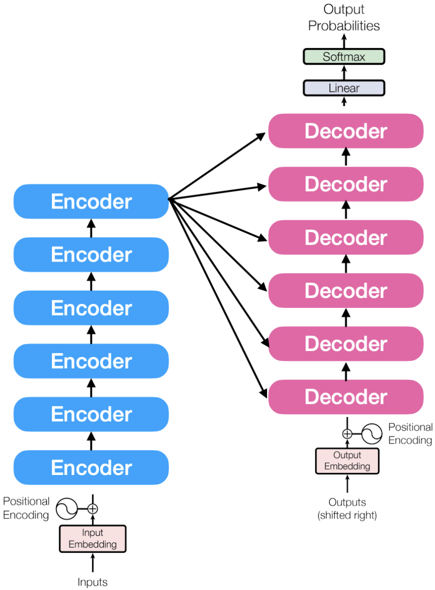


Cross-Attention

- Users previous output as query (autoregressive)
- Input as context

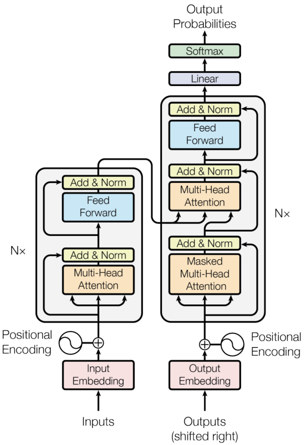


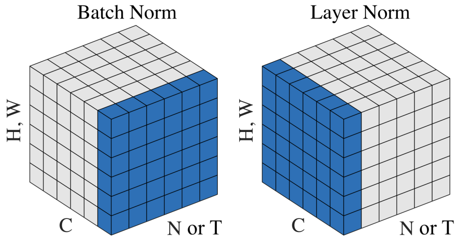
Going Deep



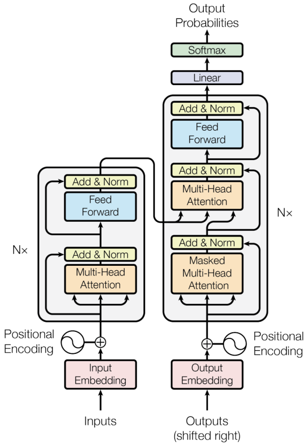
Output
Probabilities
Softmax
Linear
Decoder
Decoder
Decoder
Decoder
Decoder
Decoder
Encoder
Encoder
Encoder
Encoder
Encoder
Encoder
Positional
Encoding
Output
Embedding
Outputs
(shifted right)
Positional
Encoding
Input
Embedding
Inputs

Layer Normalization



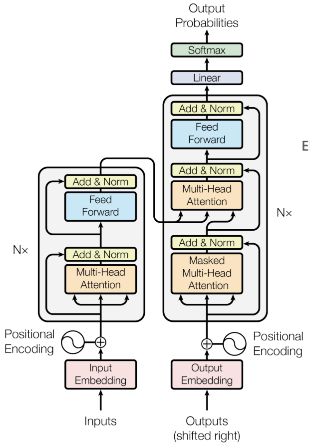


Positional Encoding

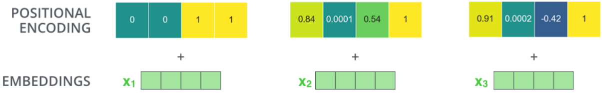


- "John loves Mary" $\neq$ "Mary loves John"
- So far, augmented context does ***not*** change when the input words change order

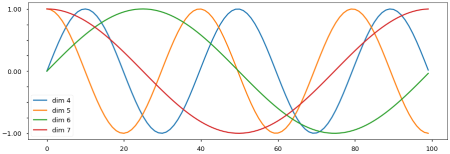
Positional Encoding



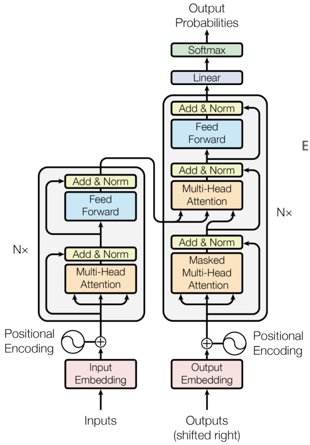
- "John loves Mary" $\neq$ "Mary loves John"
- So far, augmented context does ***not*** change when the input words change order



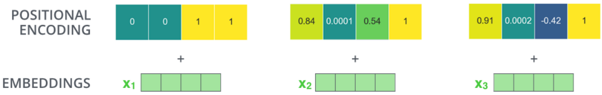
- Each word position corresponds to multiple $\sin(\cdot)$ / $\cos(\cdot)$ values of different periods



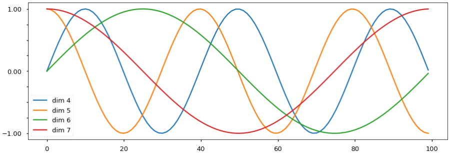
Positional Encoding



- "John loves Mary" $\neq$ "Mary loves John"
- So far, augmented context does ***not*** change when the input words change order



- Each word position corresponds to multiple $\sin(\cdot)$ / $\cos(\cdot)$ values of different periods
  - Why not linear values (e.g, 1, 2, 3, ...)?



RNNs vs. CNNs vs. Transformers

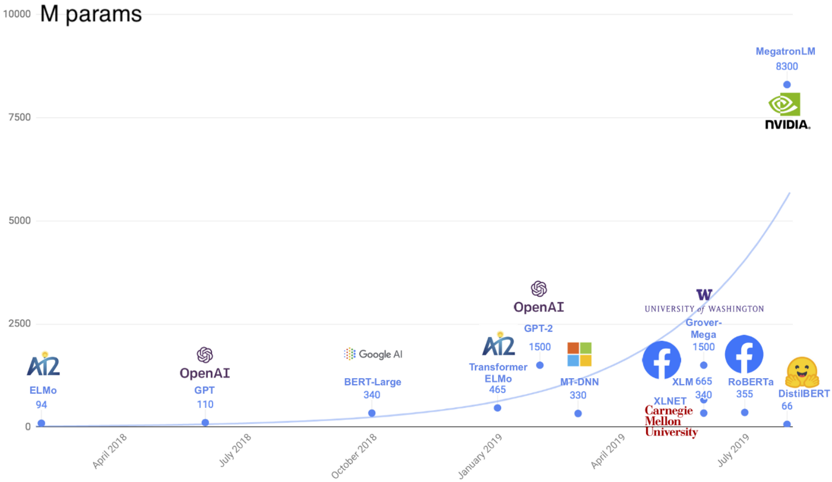
- On processing a sequence of length T at each layer with
  - D-dimensional point input and output
  - F = the CNN filter/kernel size
  - #CNN filters = D
  - #attention heads = H
  - Query, key, and value size = $\frac{D}{H}$

	#Weights	Computation	Auto-reg.	Point Dist.
CNN	$O(FD^2)$	$O(TFD^2)$	No	$O(\frac{T}{F})$
RNN	$O(D^2)$	$O(TD^2)$	Yes	$O(T)$
Self-attention	$O(D^2)$	$O(TD^2 + T^2D)$	No	$O(1)$

Outline

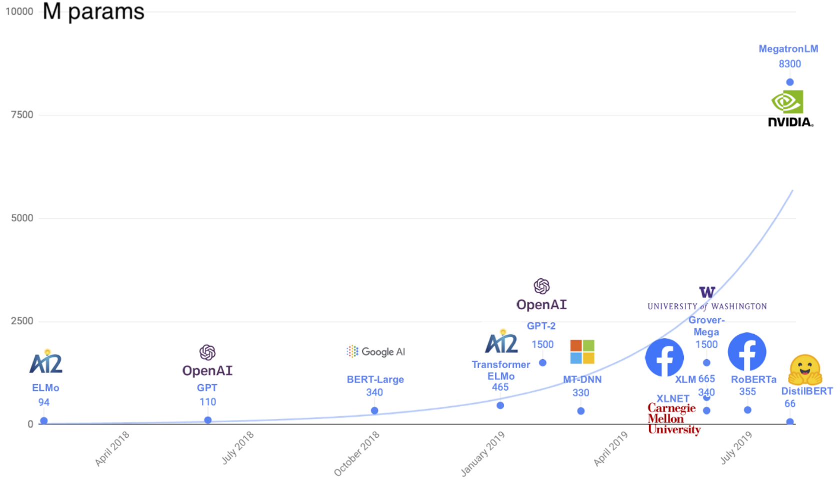
Recent Growth of Language Transformers

- All based on ***self-supervised pre-training***



Recent Growth of Language Transformers

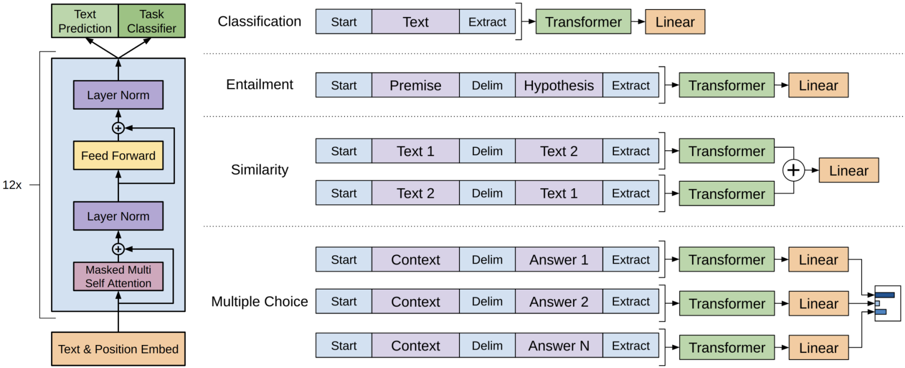
- All based on ***self-supervised pre-training***
- Costs [17]: \$2.5k–\$50k (110M params), \$10k–\$200k (340M), \$80k–\$1.6m (1.5B)



GPTv1 [12]

- 2-step training process:
  1. Generative pre-training on unlabeled data
     - to predict next word in a sentence (language model)
  2. Discriminative fine-tuning on labeled data
- Architecture based on the decoder of original transformer ("Attention is All You Need")
  - Autoregressive

Architecture & Fine-tuning Tasks



- Fine-tuning tasks: whether to language model (LM) loss or not?

Ablation Study

Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	**70.3**	**81.8**	**88.1**	**56.0**
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	**75.0**	**47.9**	**92.0**	**84.9**	**83.2**	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

BERT [4]

- Architecture based on the encoder of original transformer ("Attention is All You Need")
  - Non-autoregressive
- Pre-training tasks:
  - Masked language model ("Cloze" task)
    - "A quick brown [MASK] jumps over the lazy dog" → "fox" 11%, "ant" 5%, ...
  - Next sentence prediction
    - "[MASK] go to store [SEP] to buy a [MASK] of milk"→ True 93%, False 7%

Pre-training & Fine-tuning Tasks

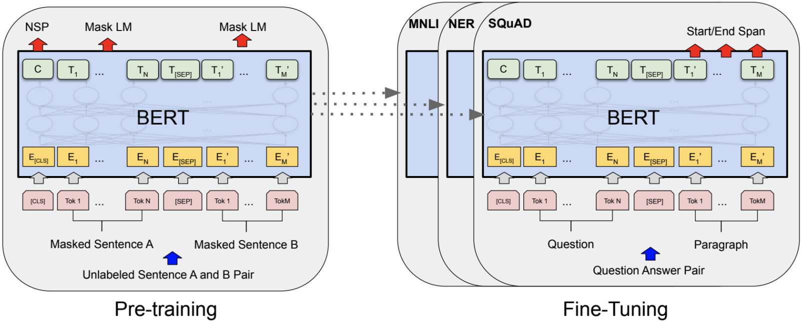


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. `[CLS]` is a special symbol added in front of every input example, and `[SEP]` is a special separator token (e.g. separating questions/answers).

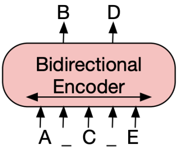
BART [11]

- BERT: bidirectional, non-autoregressive (autoencoder like)
  - Good for embedding tasks
- GPT: unidirectional, autoregressive
  - Good for generative tasks

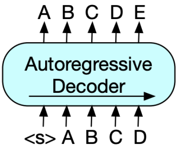
BART [11]

- BERT: bidirectional, non-autoregressive (autoencoder like)
  - Good for embedding tasks
- GPT: unidirectional, autoregressive
  - Good for generative tasks
- BART: why not combine both?

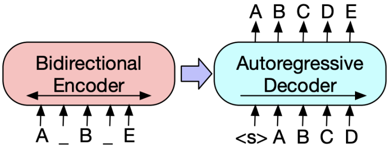
Architecture



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.

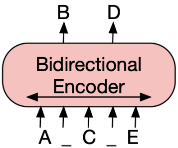


(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.

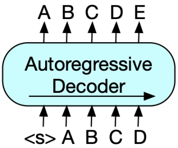


(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

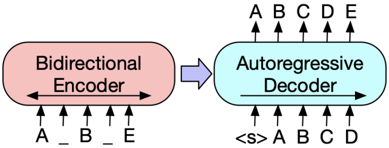
Architecture



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.



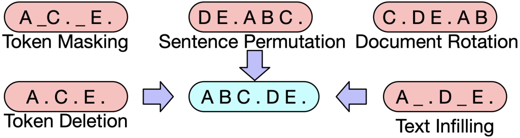
(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

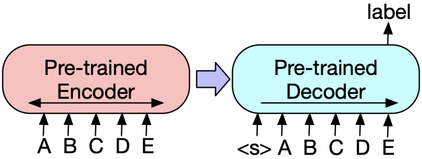
- Isn't it "Attention is All You Need?"

Denoising S2S Pre-training

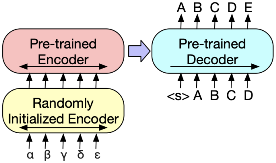


- Similar to denoising autoencoder

Fine-tuning Tasks



(a) To use BART for classification problems, the same input is fed into the encoder and decoder, and the representation from the final output is used.



(b) For machine translation, we learn a small additional encoder that replaces the word embeddings in BART. The new encoder can use a disjoint vocabulary.

GPTv2 [13] & GPTv3 [3]

- Still autoregressive
- But focus on ***transfer*** and ***few-shot learning*** scenarios
    - No need for fine-tuning ▸ Demo 1 and ▸ Demo 2
- How?

GPTv2 [13] & GPTv3 [3]

- Still autoregressive
- But focus on ***transfer*** and ***few-shot learning*** scenarios
  - No need for fine-tuning ▸ Demo 1 and ▸ Demo 2
- How? ***Multitask Language Model***
- Pre-training: to maximize $\Pr(\text{output}|\text{input}, \text{task})$ in multiple LM tasks
  - Translation: $\Pr(\text{french text}|\text{en text}, \text{translation})$
  - Question answering: $\Pr(\text{answer}|\text{question}, \text{qa})$
  - Reading comprehension: $\Pr(\text{answer}|\text{document}, \text{question}, \text{reading})$

Implicit Task Learning

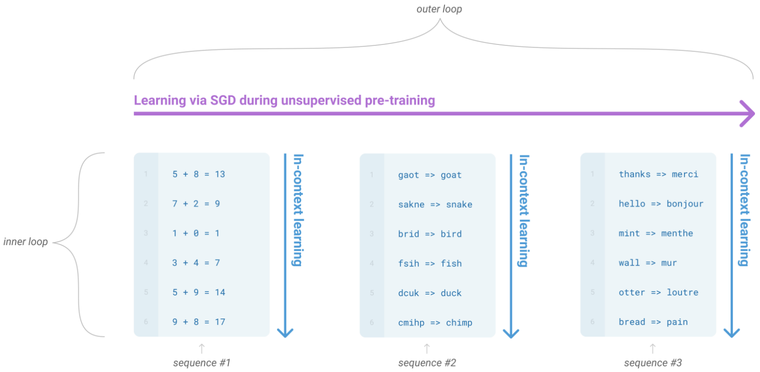
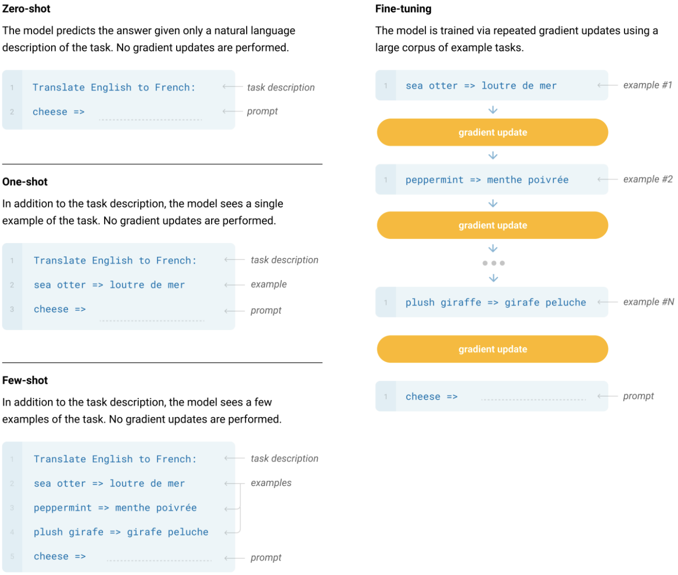


Figure 1.1: Language model meta-learning. During unsupervised pre-training, a language model develops a broad set of skills and pattern recognition abilities. It then uses these abilities at inference time to rapidly adapt to or recognize the desired task. We use the term "in-context learning" to describe the inner loop of this process, which occurs within the forward-pass upon each sequence. The sequences in this diagram are not intended to be representative of the data a model would see during pre-training, but are intended to show that there are sometimes repeated sub-tasks embedded within a single sequence.

Zero/Few Shot Learning vs. Fine-tuning



Performance vs. Model Size

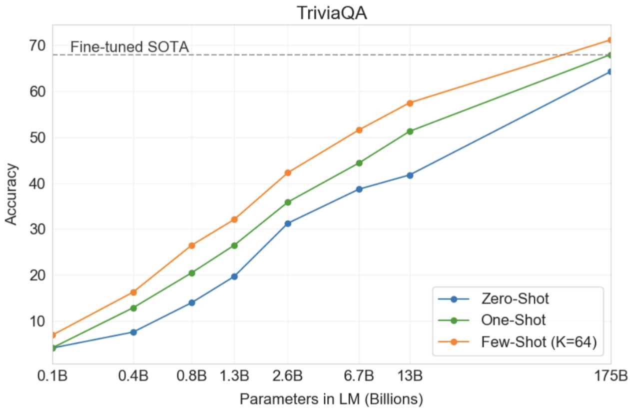
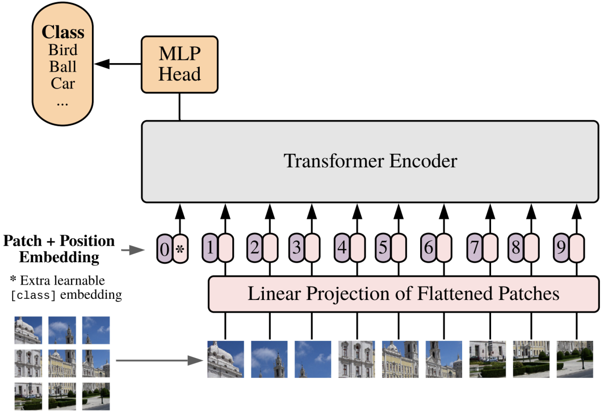


Figure 3.3: On TriviaQA GPT3's performance grows smoothly with model size, suggesting that language models continue to absorb knowledge as their capacity increases. One-shot and few-shot performance make significant gains over zero-shot behavior, matching and exceeding the performance of the SOTA fine-tuned open-domain model, RAG [LPP+20]

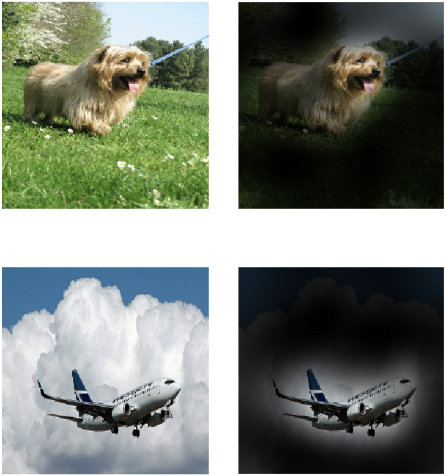
Outline

Vision Transformers (ViT) [5]

- Splits an image into 16×16 patches (words)
- Like BERT, uses [CLS] input word to get class predictions

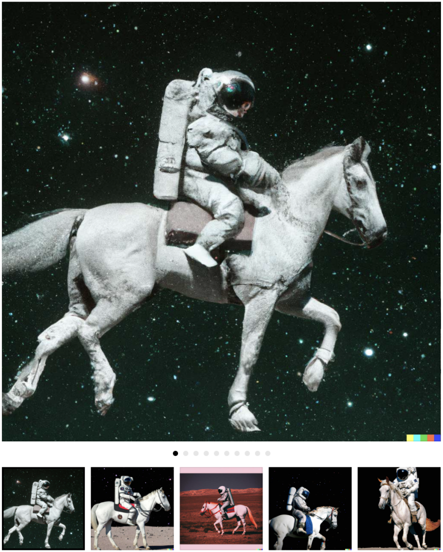


Attention



DALL-E [14]

- "An astronaut riding a horse in a photorealistic style" ▸ Live Demo



Outline

Sequence Tokenization

- Word-level
  - High input/output dimension (D)
  - Out-of-vocabulary (OOV) problem
  - "old" $\neq$ "older" $\neq$ "oldest"
- Char-level
  - Long sequence (T)

Sequence Tokenization

- Word-level
  - High input/output dimension (D)
  - Out-of-vocabulary (OOV) problem
  - "old" $\neq$ "older" $\neq$ "oldest"
- Char-level
  - Long sequence (T)
- Subword-level?
  - How to deal with OOV problem?
  - How to support different languages?

Byte Pair Encoding (BPE) [16]

- Given a training set of words
  1. Uses Unicode bytes as base symbols $\{\boldsymbol{s}^{(1)}, \boldsymbol{s}^{(2)}, \cdots\}$
  2. Merge two symbol $\boldsymbol{s}^{(i)}$ and $\boldsymbol{s}^{(j)}$ having highest $\Pr(\boldsymbol{s}^{(i)})$ and $\Pr(\boldsymbol{s}^{(j)})$
  3. Repeat step 2 until target #symbols is met

	Vocabulary	Encoded Sentence
Initialization	['a', 'c', 'b', 'e', 'i', '</w>', 'k', 'm', 'o', 'n', 'p', 's', 'r', 'u', 't', 'v', 'x']	v i e t n a m </w> t a k e s </w> m e a s u r e s </w> t o </w> b o o s t </w> r i c e </w> e x p o r t s </w>
After 1 merge operation	['a', 'c', 'b', 'e', 'p', '</w>', 'k', 'm', 'o', 'n', 'i', 's', 'r', 'u', 't', 'v', 'x', 's</w>']	v i e t n a m </w> t a k e s</w> m e a s u r e s</w> t o </w> b o o s t </w>r i c e </w> e x p o r t s</w>
After 10 merge operations	['</w>', 'vi', 'as', 'es</w>', 's</w>', 'nam', 'to', 'ri', 't</w>', 'ort', 'a', 'c', 'b', 'e', 'k', 'm', 'o', 'p', 's', 'r', 'u', 't', 'x']	vi e t nam </w> t a k es</w> m e as u r es</w> to </w> b o o s t</w> ri c e </w> e x p ort s</w>
After 34 merge operations	['takes</w>', 'measures</w>', 'exports</w>', 'boost</w>', 'rice</w>', 'vietnam</w>', 'to</w>']	vietnam</w> takes</w> measures</w> to</w> boost</w> rice</w> exports</w>

Other Variants

- BPE is used by GPT
- WordPiece [15]:
    - Merge $\boldsymbol{s}^{(i)}$ and $\boldsymbol{s}^{(j)}$ having highest $\frac{\Pr(\boldsymbol{s}^{(i)},\boldsymbol{s}^{(j)})}{\Pr(\boldsymbol{s}^{(i)})\Pr(\boldsymbol{s}^{(j)})}$
    - Used by BERT
- Unigram Language Model [8]
    - Top-down, probabilistic
- SentencePiece [9]
    - Treat space "_" as symbols to support difference languages
    - Merge algorithm: BPE or Unigram Language Model
    - Used by ALBERT, XLNet, Marian, T5

Reference I

[1] Martin Arjovsky, Amar Shah, and Yoshua Bengio.
Unitary evolution recurrent neural networks.
arXiv preprint arXiv:1511.06464, 2015.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio.
Neural machine translation by jointly learning to align and translate.
arXiv preprint arXiv:1409.0473, 2014.

[3] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al.
Language models are few-shot learners.
Advances in neural information processing systems, 33:1877–1901, 2020.

Reference II

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova.
Bert: Pre-training of deep bidirectional transformers for language understanding.
arXiv preprint arXiv:1810.04805, 2018.

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al.
An image is worth 16x16 words: Transformers for image recognition at scale.
arXiv preprint arXiv:2010.11929, 2020.

[6] Andrej Karpathy, Justin Johnson, and Li Fei-Fei.
Visualizing and understanding recurrent networks.
arXiv preprint arXiv:1506.02078, 2015.

Reference III

[7] David Krueger and Roland Memisevic.
Regularizing rnns by stabilizing activations.
arXiv preprint arXiv:1511.08400, 2015.

[8] Taku Kudo.
Subword regularization: Improving neural network translation models with multiple subword candidates.
arXiv preprint arXiv:1804.10959, 2018.

[9] Taku Kudo and John Richardson.
Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing.
arXiv preprint arXiv:1808.06226, 2018.

[10] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton.
A simple way to initialize recurrent networks of rectified linear units.
arXiv preprint arXiv:1504.00941, 2015.

Reference IV

[11] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer.
Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.
arXiv preprint arXiv:1910.13461, 2019.

[12] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al.
Improving language understanding by generative pre-training.
OpenAI blog, 2018.

[13] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al.
Language models are unsupervised multitask learners.
OpenAI blog, 2019.

Reference V

[14] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen.
Hierarchical text-conditional image generation with clip latents.
arXiv preprint arXiv:2204.06125, 2022.

[15] Mike Schuster and Kaisuke Nakajima.
Japanese and korean voice search.
In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[16] Rico Sennrich, Barry Haddow, and Alexandra Birch.
Neural machine translation of rare words with subword units.
arXiv preprint arXiv:1508.07909, 2015.

[17] Or Sharir, Barak Peleg, and Yoav Shoham.
The cost of training nlp models: A concise overview.
arXiv preprint arXiv:2004.08900, 2020.

Reference VI

[18] Eduardo D Sontag.
On the computational power of neural nets'.
JOURNAL OF COMPUTER AND SYSTEM SCIENCES, 50:132–150, 1995.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
Advances in neural information processing systems, 30, 2017.

[20] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S Zemel, and Yoshua Bengio.
Show, attend and tell: Neural image caption generation with visual attention.
arXiv preprint arXiv:1502.03044, 2(3):5, 2015.