TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2019

Language Modeling

Machine Translation

Attention

Beam Search

Error-Based Training

Digression on Tensor Notation

We will assume a high level RNN procedure RNN_Φ which takes a sequence of input vectors and returns a sequence of hidden state vectors.

$$h[T,J] = \mathrm{RNN}_\Phi(x[T,I])$$

Here, and in the future, we use capital letter indeces to denote tensors and lower case indeces to denote particular values from tensors.

We will also omit the batch index. Minibatching can be viewed as an efficiency optimization.

Digression on Tensor Notation

We use capital letter indeces to denote tensors and lower case indeces to denote particular values from tensors.

The equation

$$\tilde{L}[\ell+1,j] = \left(\sum_i W[\ell,i,j]L[\ell,i]\right) + \beta[\ell,j]$$

stands for a set of assignments — one for each value ℓ and j

Slicing Notation

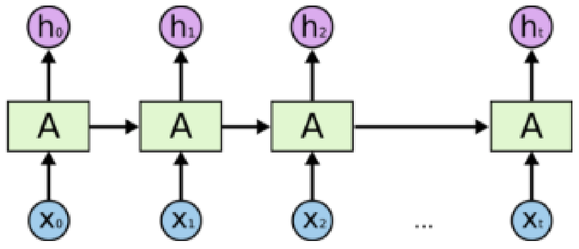
A mixture of upper and lower case indeces denotes a slice.

$h[t, J]$ denotes the vector whose jth component is $h[t, j]$.

$L[X, Y, j]$ denotes the matrix (image) determined by the jth channel.

Different Forms of RNN

Basic RNN



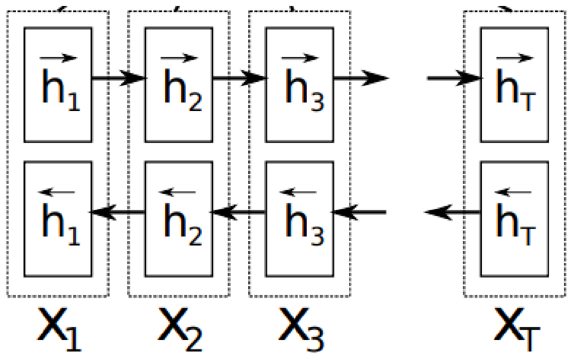
[Christopher Olah]

Procedure $\mathrm{RNN}_{\Phi}(x(T, I))$

$$h[t, J] \;=\; \mathrm{CELL}_{\Phi.\mathrm{cell}}(\mathrm{if}(t = 0, \Phi.\mathrm{init}[J], h[t-1, J]),\; x[t, I])$$

Return $h[T, J]$

bi-directional RNNS

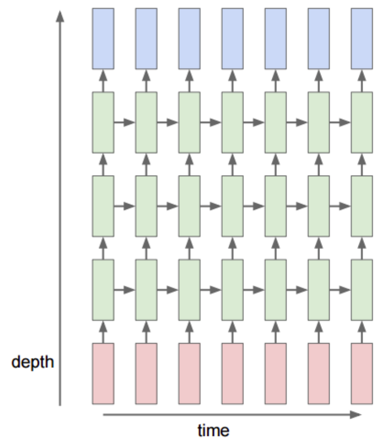


$$\vec{h}[T,\vec{J}] = \mathrm{R\vec{N}N}_{\Phi.LR}(x[T,I])$$

$$\overleftarrow{h}[T,\overleftarrow{J}] = \mathrm{R\overleftarrow{N}N}_{\Phi.RL}(x[T,I])$$

$$h[t,J] = [\vec{h}[T,\vec{J}],\ \overleftarrow{h}[t,\overleftarrow{J}]] \text{ where } [x,y] \text{ is vector concatenation}$$

Multi-Layer RNNs



[Figure by Leonardo Araujo dos Santos]

$$h[0, T, J] \;=\; \mathrm{RNN}_{\Phi.0}(x[T, I])$$

$$h[\ell+1, T, J] \;=\; \mathrm{RNN}_{\Phi.\ell+1}(h[\ell, T, J])$$

Each layer can be bidirectional.

Residual Multi-Layer RNNs

$$h[0,T,J] \;=\; \mathrm{RNN}_{\Phi.0}(x[T,I])$$

$$h[\ell+1,T,J] \;=\; h[\ell,T,J] + \mathrm{RNN}_{\Phi.\ell+1}(h[\ell,T,J])$$

Each layer can be bidirectional.

This is used in Google translation.

Language Modeling

Let W be some finite vocabulary of tokens (words).

Let Pop be a population distribution over W^* (sentences).

We will write a sequence $w[1], \ldots, w[t]$ as $w[T]$.

We want to train a model $P_\Phi(w[T])$.

$$\Phi^* = \operatorname*{argmin}_\Phi \; E_{\mathrm{Pop}} \; - \ln P_\Phi(w[T])$$

The End of Sentence Token

We assume a special toeken `<EOS>` called the end of sentence token.

Let t_{final} be the last time index allowed for T.

We requite $w[t_{\mathrm{final}}] = \texttt{<EOS>}$ and $w[t] \neq \texttt{<EOS>}$ for $t < t_{\mathrm{final}}$.

This gives:

$$P(w[T]) = \prod_t P(w[t] \mid w[1], \ldots, w[t-1])$$

Word Embeddings

We will use a word embedding tensor $e[W, I]$ which should be interpreted as assigning a vector $e[w, I]$ to each word w.

The word embedding tensor $e[W, I]$ is a parameter of language models (and many other kinds of NLP models).

Autoregressive Language Modeling

Procedure $P_\Phi(w[T])$

$$\begin{aligned} x[t,I] &= (\Phi.\mathrm{embed})[w[t],I] \\ h[T,J] &= \mathrm{RNN}_{\Phi.\mathrm{RNN}}(x[T,I]) \\ s[t,w] &= \sum_{i,j} (\Phi.\mathrm{embed})[w,i]\ \ (\Phi.\mathrm{score})[i,j]\ \ \mathrm{if}(t=0, \Phi.\mathrm{init}[j], h[t-1,j]) \\ p[t,w] &= \operatorname*{softmax}_w\ s[t,w] \end{aligned}$$

Return $\prod_t p[t, w[t]]$

Language Model Loss Decomposition

$$\Phi^* = \operatorname*{argmin}_{\Phi} \; E_{\mathrm{Pop}} \; -\ln \; P_\Phi(w[T])$$

$$= \operatorname*{argmin}_{\Phi} \; E_{\mathrm{Pop}} \; \sum_t \; -\ln \; p[t, w[t]]$$

Standard Measures of Performance

Bits per Character: For character language models performance is measured in bits per character. Typical numbers are slightly over one bit per character.

Perplexity: It would be natural to measure word language models in bits per word. However, it is traditional to measure then in perplexity which is defined to be 2^b where b is bits per word. Perplexities of about 60 are typical.

According to Quora there are 4.79 letters per word. 1 bit per character (including space characters) gives a perplexity of $2^{5.79}$ or 55.3.

Sampling From an Autoregressive Model

To sample a sentence

$$w_1, \ldots, w_T, \texttt{<eos>}$$

we sample w_t from

$$P_\Phi(w_t | w_1, \ldots, w_{t-1})$$

until we get `<eos>`.

Machine Translation

$$w_1^{\mathrm{in}}, \ldots, w_{t_{\mathrm{in}}}^{\mathrm{in}} \Rightarrow w_1^{\mathrm{out}}, \ldots, w_{t_{\mathrm{out}}}^{\mathrm{out}}$$

$$w_{\mathrm{in}}[T_{\mathrm{in}}] \Rightarrow w_{\mathrm{out}}[T_{\mathrm{out}}]$$

Translation is a **sequence to sequence** (seq2seq) task.

Sequence to Sequence Learning with Neural Networks, Sutskever, Vinyals and Le, NIPS 2014, arXiv Sept 10, 2014.

Machine Translation

$$w_{\mathrm{in}}[T_{\mathrm{in}}] \Rightarrow w_{\mathrm{out}}[T_{\mathrm{out}}]$$

We define a model

$$P_\Phi\left(w_{\mathrm{out}}[T_{\mathrm{out}}] \mid w_{\mathrm{in}}[T_{\mathrm{in}}]\right)$$

$$\begin{aligned}\Phi^* &= \underset{\Phi}{\mathrm{argmin}}\;\; E_{\mathrm{Pop}}\;\; -\ln\;\; P_\Phi(w_{\mathrm{out}}[T_{\mathrm{out}}] \mid w_{\mathrm{in}}[T_{\mathrm{in}}]) \\ &= \underset{\Phi}{\mathrm{argmin}}\;\; E_{\mathrm{Pop}}\;\; -\ln P_\Phi(y|x)\end{aligned}$$

A Simple RNN Translation Model

We construct a conditional languge model

$$P_\Phi(w_{\mathrm{out}}[T_{\mathrm{out}}] \mid w_{\mathrm{in}}[T_{\mathrm{in}}]) = P_{\Phi.\mathrm{out}}(w_{\mathrm{out}}[T_{\mathrm{out}}] \mid H_{\Phi.\mathrm{in}}(w_{\mathrm{in}}[T_{\mathrm{in}}]))$$

Here $H_{\Phi.\mathrm{in}}(w_{\mathrm{in}}[T_{\mathrm{in}}])$ is used as the initial hidden state of an RNN lnguage model.

The initial hidden state $H_{\Phi.\mathrm{in}}(w_{\mathrm{in}}[T_{\mathrm{in}}])$ is a “thought vector” representation of the input sentence.

Computing a Thought Vector

The thought vector for a sentence can be taken to be the final hidden state of a right-to-left RRN run on the sentence.

```
Procedure $H_\Phi(w[T])$
    $\overleftarrow{h}[T,J] = \mathrm{R}\overleftarrow{\mathrm{N}}\mathrm{N}_\Phi\,(w[T])$
Return $\overleftarrow{h}[t_{\mathrm{init}}, J]$
```

For a bidirectional RNN the thought vector is typically

$$\left[\overleftarrow{h}_{\mathrm{in}}[t_{\mathrm{init}}, \overleftarrow{J}],\ \vec{h}_{\mathrm{in}}[t_{\mathrm{final}}, \vec{J}]\right]$$

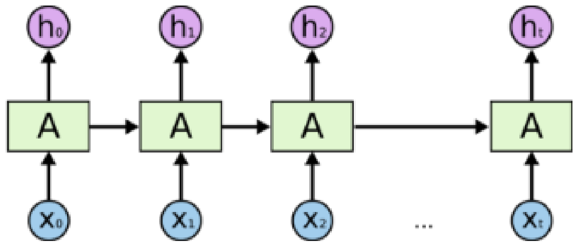
A Conditional RNN Language Modeling

Procedure $P_\Phi(w[T] \mid H[J])$

$$
\begin{aligned}
x[t, I] &= (\Phi.\mathrm{embed})[w[t], I] \\
h[T, J] &= \mathrm{RNN}_{\Phi.\mathrm{RNN}}(x[T, I] \mid H[J]) \\
s[t, w] &= \sum_{i,j} (\Phi.\mathrm{embed})[w, i] \;\; (\Phi.\mathrm{score})[i, j] \;\; \mathrm{if}(t = 0, H[j], h[t-1, j]) \\
p[t, w] &= \operatorname*{softmax}_{w} \; s[t, w]
\end{aligned}
$$

Return $\prod_t p[t, w[t]]$

A Conditional RNN



[Christopher Olah]

Procedure $\mathrm{RNN}_\Phi(x(T, I) \mid H[J])$

$$h[t, J] = \mathrm{CELL}_\Phi(\mathrm{if}(t = 0, H[J], h[t-1, J]),\ x[t, I])$$

Return $h[T, J]$

Machine Translation Decoding

We can sample from $P_{\Phi.\mathrm{out}}(w[T] \mid H[J])$.

But we might prefer

$$w_{\mathrm{out}}[T_{\mathrm{out}}] = \operatorname*{argmax}_{w_{\mathrm{out}}[T_{\mathrm{out}}]} P_{\Phi}\left(w_{\mathrm{out}}[T_{\mathrm{out}}] \mid w_{\mathrm{in}}[T_{\mathrm{in}}]\right)$$

This is typically approximated with greedy decoding:

$$w_{\mathrm{out}}[t+1] = \operatorname*{argmax}_{w} p_{\mathrm{out}}[t+1, w]$$

These are not the same.

Attention-Based Translation

Neural Machine Translation by Jointly Learning to Align and Translate Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio, ICLR 2015 (arXiv Sept. 1, 2014)

Thought Vectors Augmented by Structed Memory

The input sentence is now represented by both a thought vector and structured memory (a sequence of vectors).

$$P_\Phi(w_{\mathrm{out}}[T_{\mathrm{out}}] \mid w_{\mathrm{in}}[T_{\mathrm{in}}])$$

$$= P_{\Phi.\mathrm{out}}(w_{\mathrm{out}}[T_{\mathrm{out}}] \mid H_{\Phi.H}(w_{\mathrm{in}}[T_{\mathrm{in}}]),\ \mathrm{RNN}_{\Phi.\mathrm{RNN}}(w_{\mathrm{in}}[T_{\mathrm{in}}]))$$

Attention-Based Translation

Procedure $P_\Phi(w[T] \mid H[J],\ M[T_M, J])$

$$x[t, I] = (\Phi.\text{embed})[w[t], I]$$

$$h[T, J] = \text{RNN}_{\Phi.\text{RNN}}(x[T, I] \mid H[J],\ M[T_M, J])$$

$$s[t, w] = \sum_{i,j} (\Phi.\text{embed})[w, i]\ \ (\Phi.\text{score})[i, j]\ \ \text{if}(t = 0,\ H[j],\ h[t-1, j])$$

$$p[t, w] = \underset{w}{\text{softmax}}\ s[t, w]$$

Return $\prod_t\ p[t, w[t]]$

Attention-Based Conditional RNN

Procedure $\mathrm{RNN}_\Phi(x[T_x, I] \mid H[J],\ M[T_M, J])$

$$h[t, J] = \begin{cases} \mathrm{CELL}_\Phi(H[J], [x[t, I], \mathrm{Val}(H[J], M[T, J])]) \text{ if } t = t_{\mathrm{init}} \\ \mathrm{CELL}_\Phi(h[t-1, J], [x[t, I], \mathrm{Val}(h[t-1, J], M[T, J])]) \text{ o.w.} \end{cases}$$

Return $h[T, J]$

Attention as a Key-Value Memory Mechanism

Procedure Val(key$[J]$, $M[T, J]$)

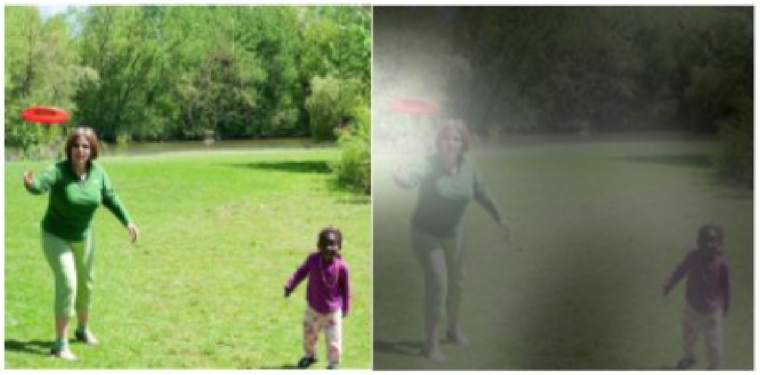
$$s[t] = \mathrm{key}[J]^\top M[t, J]$$

$$\alpha[t] = \operatorname*{softmax}_t \; s[t] \text{ ; the attention}$$

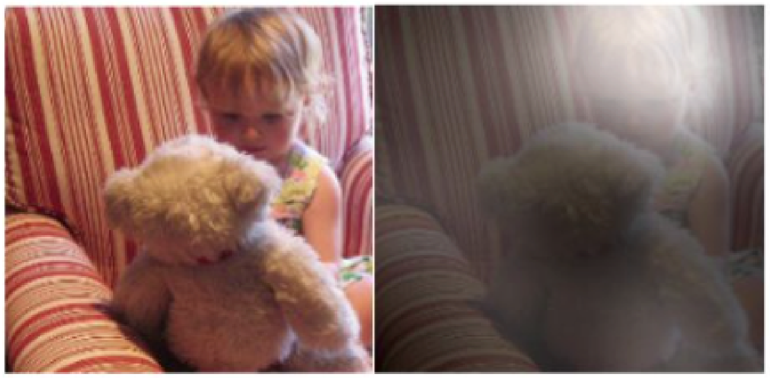
$$V[J] = \sum_t \alpha[t] M[t, J]$$

Return $V[J]$

Attention in Image Captioning



A woman is throwing a frisbee in a park.



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

Xu et al. ICML 2015

Greedy Decoding vs. Beam Search

We would like

$$W_{\mathrm{out}}[T_{\mathrm{out}}]^* = \operatorname*{argmax}_{W_{\mathrm{out}}[T_{\mathrm{out}}]} \; P_\Phi(W_{\mathrm{out}}[T_{\mathrm{out}}] \mid W_{\mathrm{in}}[T_{\mathrm{in}}])$$

But a greedy algorithm may do well

$$w_t = \operatorname*{argmax}_{w} \; P_\Phi(w \mid W_{\mathrm{in}}[T_{\mathrm{in}}],\; w_1, \ldots, w_{t-1})$$

But these are not the same.

Example

"Those apples are good" vs. "Apples are good"

$$P_\Phi(\text{Apples are Good } \texttt{<eos>}) > P_\Phi(\text{Those apples are good } \texttt{<eos>})$$

$$P_\Phi(\text{Those}|\varepsilon) > P_\Phi(\text{Apples}|\varepsilon)$$

Beam Search

At each time step we maintain a list the K best words and their associated hidden vectors.

This can be used to produce a list of k “best” decodings which can then be compared to select the most likely one.

Phrase Based Statistical Machine Translation (SMT)

Step I: Learn a phrase table — a set of triples (p, q, s) where

- p is a (short) sequence of source words.
- q is a (short) sequence of target words.
- s is a score.

(“au”, “to the”, .5)   (“au banque”, “for the bank”, .01)

For a phrase triple P we will write P.source for the source phrase, P.target for the target phrase, and P.score for the score.

Derivations

Consider an input sentence x of length T.

We will write $x[s:t]$ for the substring $x[s]$, …, $x[t-1]$.

A derivation d from x is a sequence $(P_1, s_1, t_1,)$, …, (P_K, s_K, t_K) where P_k.source $= x[s_k : t_k]$.

The substrings $x[s_k : t_k]$ should be disjoint and "cover" x.

For $d = [(P_1, s_1, t_1,)$, …, $(P_L, s_K, t_K)]$ we define

$$y(d) \equiv P_1.\text{target} \;\cdots P_K.\text{target}$$

We let $D(x)$ be the set of derivations from x.

Scoring

For $d \in D(x)$ we define a score $s(d)$

$$s(d) = \alpha \ln P_{\mathrm{LM}}(y(d)) + \beta \sum_k P_k.\mathrm{score} + \gamma \ \mathrm{distortion}(d)$$

where $P_{\mathrm{LM}}(y)$ is the probability assigned to string y under a language model for the target language

and distortion(d) is a measure of consistency of word ordering between source and target strings as defined by the indeces (s_1, t_1), ..., (s_K, t_K).

Translation

$$y(x) = y(d^*(x))$$

$$d^*(x) = \operatorname*{argmax}_{d \in D(x)} s(d)$$

END