DNN, CNN, RNN, LSTM, Attn, and Transformer

Shoichiro Yamanishi

May 29, 2020

1 Introduction

This document describes the following.

- DNN Backprop mechanism
- CNN Forward propagation and Backprop in multi-channel 2-dimensional convolution kernel with step size s.
- RNN Backprop
- The reason of LSTM as a generalization of leaky units with learned parameters to cope with vanishing gradient problem.
- Traditional Attention on top of bi-directional RNN
- Transformer's multi-head attention part.

2 Backprop : DNN

The fully connected deep neural network is defined as follows.

$$\begin{aligned}
&\boldsymbol{x} : \text{sample input}\\
&\boldsymbol{y} : \text{sample output}\\
&W^{(i)} : \text{weights to be leaned}\\
&\boldsymbol{b}^{(i)} : \text{bias to be leaned}\\
&\boldsymbol{h}^{(0)} = \boldsymbol{x} \qquad (1)\\
&\boldsymbol{a}^{(i)} = W^{(i)}\boldsymbol{h}^{(i-1)} + \boldsymbol{b}^{(i)} : \text{activation at the i-th layer}\\
&\boldsymbol{h}^{(i)} = \boldsymbol{f}^{(i)}(\boldsymbol{a}^{(i)}) : \text{hidden variable at the i-th layer}\\
&\hat{\boldsymbol{y}} = \boldsymbol{h}^{(l)} : \text{output}\\
&J(\hat{\boldsymbol{y}}, \boldsymbol{y}) : \text{loss function}
\end{aligned}$$

The back propagation (of the derivatives of the loss function) is derived as follows. First, the top level:

$$\frac{\partial J}{\partial \boldsymbol{h}^{(l)}} = \frac{\partial J}{\partial \hat{\boldsymbol{y}}} \tag{2}$$

Equation 2 corresponds to the first assignment to $\boldsymbol{g}$ in Algorithm 6.4 in pp 206 of DL [2].

$$\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{a}^{(l)}} &= \frac{\partial J}{\partial \boldsymbol{h}^{(l)}} \frac{\partial \boldsymbol{h}^{(l)}}{\partial \boldsymbol{a}^{(l)}} = \frac{\partial J}{\partial \hat{\boldsymbol{y}}} \boldsymbol{f}^{(l)\prime} \\
\frac{\partial J}{\partial W^{(l)}} &= \frac{\partial J}{\partial \hat{\boldsymbol{y}}} \frac{\partial \boldsymbol{h}^{(l)}}{\partial \boldsymbol{a}^{(l)}} \frac{\partial \boldsymbol{a}^{(l)}}{\partial W^{(l)}} = \frac{\partial J}{\partial \hat{\boldsymbol{y}}} \boldsymbol{f}^{(l)\prime} \boldsymbol{h}^{(l-1)} \\
\frac{\partial J}{\partial \boldsymbol{b}^{(l)}} &= \frac{\partial J}{\partial \hat{\boldsymbol{y}}} \frac{\partial \boldsymbol{h}^{(l)}}{\partial \boldsymbol{a}^{(l)}} \frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{b}^{(l)}} = \frac{\partial J}{\partial \hat{\boldsymbol{y}}} \boldsymbol{f}^{(l)\prime}
\end{aligned} \tag{3}$$

Next, down to the bottom level.

$$\begin{aligned}
\frac{\partial J}{\partial W^{(i)}} &= \frac{\partial J}{\partial \boldsymbol{a}^{(i)}} \frac{\partial \boldsymbol{a}^{(i)}}{\partial W^{(i)}} = \frac{\partial J}{\partial \boldsymbol{a}^{(i)}} \boldsymbol{h}^{(i-1)} \\
\frac{\partial J}{\partial \boldsymbol{b}^{(i)}} &= \frac{\partial J}{\partial \boldsymbol{a}^{(i)}} \frac{\partial \boldsymbol{a}^{(i)}}{\partial \boldsymbol{b}^{(i)}} = \frac{\partial J}{\partial \boldsymbol{a}^{(i)}}
\end{aligned} \tag{4}$$

$\frac{\partial J}{\partial \boldsymbol{a}^{(i)}}$, which corresponds to the second assignment to $\boldsymbol{g}$ in Algorithm 6.4 in pp 206 of DL [2], is recursively defined as follows.

$$\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{a}^{(i)}} &= \sum_{\boldsymbol{a}^{(i+1)}} \frac{\partial J}{\partial \boldsymbol{a}^{(i+1)}} \frac{\partial \boldsymbol{a}^{(i+1)}}{\partial \boldsymbol{a}^{(i)}} \\
&= \sum_{\boldsymbol{a}^{(i+1)}} \frac{\partial J}{\partial \boldsymbol{a}^{(i+1)}} \frac{\partial \boldsymbol{a}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}} \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{a}^{(i)}} \\
&= \left(\sum_{\boldsymbol{a}^{(i+1)}} \frac{\partial J}{\partial \boldsymbol{a}^{(i+1)}} W^{(i)} \right) \boldsymbol{f}^{(l)\prime}
\end{aligned} \tag{5}$$

where the facter inside the parenthesis corresponds to the third assignment to $\boldsymbol{g}$ in Algorithm 6.4 in pp 206 of DL [2].

For general back propagation using the computational graph used by TensorFlow is presented in figure 6.11 of DL [2].

3 CNN

Following the notational convention by DL [2], one leyar of 2-dimentional convolution layer from an input V to its activation Z with step s is defined as follows.

$$\begin{array}{llllll} V_{l,j,k} & - & \textit{input} & l:\text{i-channel} & j:\text{coord1} & k:\text{coord2} \\ K_{i,l,m,n} & - & \textit{conv kernel} & i:\text{o-channel} & l:\text{i-channel} & m:\text{coord1} \quad n:\text{coord1} \\ Z_{i,j,k} & - & \textit{activation} & i:\text{o-channel} & j:\text{coord1} & k:\text{coord2} \end{array}$$

$$Z_{i,j,k} = c\left(K,V,s\right) = \sum_{l,m,n} \left[V_{l,(j-1)s+m,(k-1)s+n} K_{i,l,m,n}\right] \quad - \quad \text{forward propagation} \tag{6}$$

V can be an image with l indexing over the RGBA channel, and j,k over the xy-coordinates. s is the step size over V. Please see figure 1 and 2.

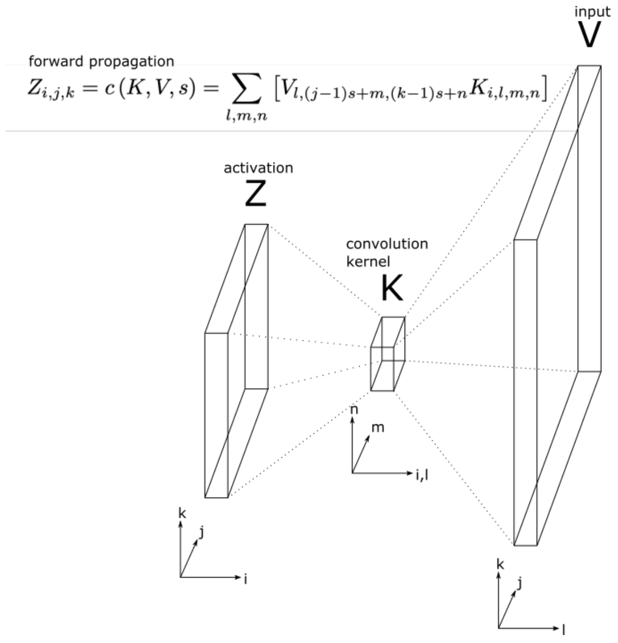

forward propagation
$Z_{i,j,k} = c\left(K,V,s\right) = \sum_{l,m,n} \left[V_{l,(j-1)s+m,(k-1)s+n} K_{i,l,m,n}\right]$
input
V
activation
Z
convolution kernel
K
n
m
i,l
k
j
i
k
j
l

Figure 1: CNN Architecture

3.1 Backprop

Assume the backprop is available up to Z as in:

$$G_{i,j,k} = \frac{\partial}{Z_{i,j,k}} J(\hat{\boldsymbol{y}}, \boldsymbol{y}) \tag{7}$$

We want to find $\frac{\partial}{K_{i,l,m,n}} J(\hat{\boldsymbol{y}}, \boldsymbol{y})$ for parameter update, and $\frac{\partial}{V_{l,j,k}} J(\hat{\boldsymbol{y}}, \boldsymbol{y})$ to back propagate the derivattive down the network.

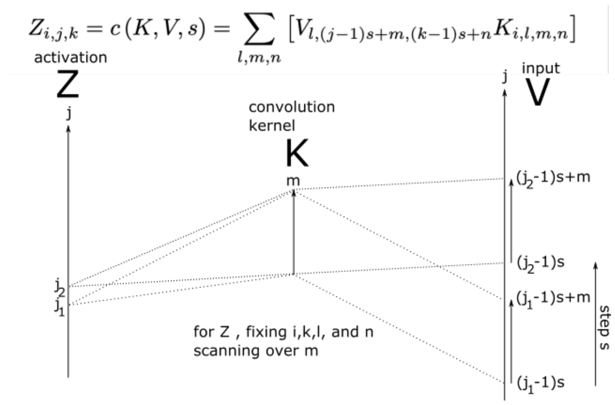

$Z_{i,j,k} = c(K,V,s) = \sum_{l,m,n} \left[V_{l,(j-1)s+m,(k-1)s+n} K_{i,l,m,n} \right]$
activation
Z
j
convolution kernel
K
m
input
V
j
$(j_2-1)s+m$
$(j_2-1)s$
$(j_1-1)s+m$
$(j_1-1)s$
j_2
j_1
step s
for Z , fixing i,k,l, and n scanning over m

Figure 2: Forward Propagation with Convolution

$$
\begin{aligned}
g(G,V,s)_{i,l,m,n} &= \frac{\partial}{\partial K_{i,l,m,n}} J(\hat{\boldsymbol{y}}, \boldsymbol{y}) \\
&= \sum \frac{\partial J}{\partial Z} \frac{\partial Z}{\partial K} \\
&= \sum_{j,k} G_{i,j,k} V_{l,(j-1)s+m,(k-1)s+n}
\end{aligned}
\tag{8}
$$

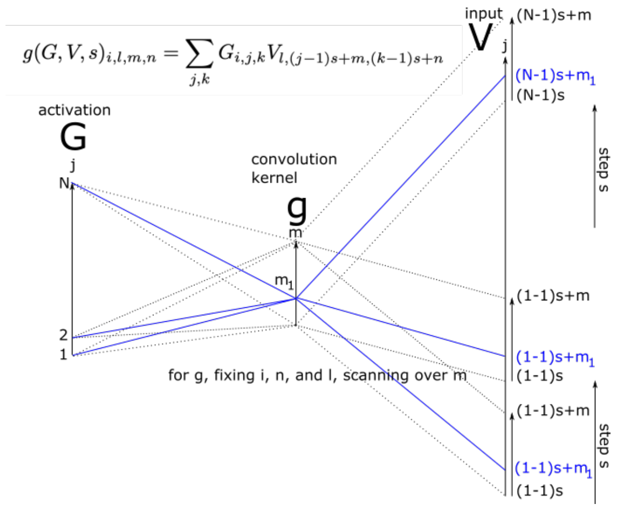


Figure 3: Backward Propagation to $g(G,V,s)_{i,l,m,n} = \frac{\partial}{\partial K_{i,l,m,n}} J(\hat{\boldsymbol{y}}, \boldsymbol{y})$

$$
\begin{aligned}
h(K,G,s)_{l,j^V,k^V} &= \frac{\partial}{\partial V_{l,j^V,k^V}} J(\hat{\boldsymbol{y}},\boldsymbol{y}) \\
&= \sum \frac{\partial J}{\partial Z}\frac{\partial Z}{\partial V} \\
&= \sum_{j^Z,m}^{s.t.(j^Z-1)s+m=j^V} \sum_{k^Z,n}^{s.t.(k^Z-1)s+n=k^V} \sum_i K_{i,l,m,n} G_{i,j^Z,k^Z}
\end{aligned} \tag{9}
$$

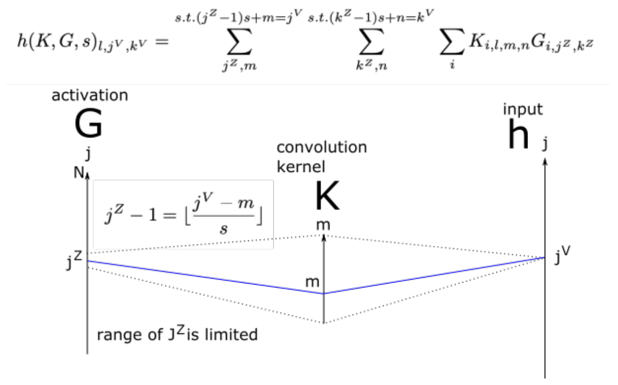

$h(K,G,s)_{l,j^V,k^V} = \sum_{j^Z,m}^{s.t.(j^Z-1)s+m=j^V} \sum_{k^Z,n}^{s.t.(k^Z-1)s+n=k^V} \sum_i K_{i,l,m,n} G_{i,j^Z,k^Z}$
activation
G
j
N
$j^Z - 1 = \lfloor \frac{j^V - m}{s} \rfloor$
convolution kernel
K
m
input
h
j
j^Z
m
j^V
range of J^Z is limited

Figure 4: Backward Propagation to $h(K,G,s)_{l,j^V,k^V} = \frac{\partial}{\partial V_{l,j^V,k^V}} J(\hat{\boldsymbol{y}},\boldsymbol{y})$

4 RNN

Here I explain a simple case where the hidden layer uses *tanh* and the final layer *softmax*.

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + W\boldsymbol{h}^{(t-1)} + U\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + V\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= softmax(\boldsymbol{o}^{(t)})
\end{aligned} \tag{10}
$$

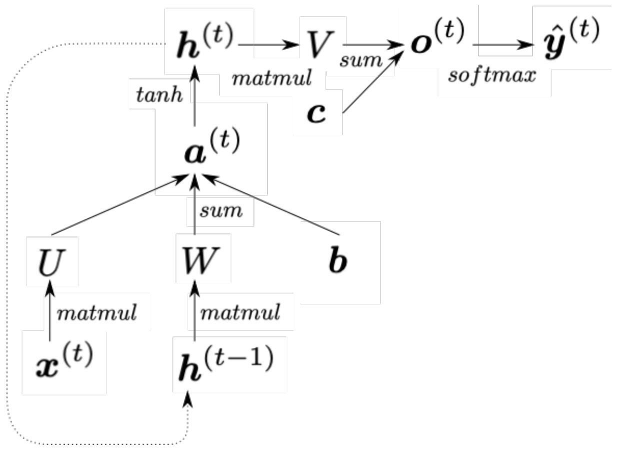


Figure 5: Typical RNN

4.1 Backprop

$$\begin{aligned}
\left[\frac{\partial J}{\partial \boldsymbol{o}^{(t)}}\right]_i &= \left[\frac{\partial softmax(\boldsymbol{o}^{(t)})}{\partial \boldsymbol{o}^{(t)}}\right]_i \\
&= \left(\delta_{i,j}^{(t)} - \frac{\exp(o_i)}{\sum_k \exp(o_k)}\right) \quad \text{where } \hat{y}_j = 1 \\
\\
\frac{\partial J}{\partial V} &= \sum_t \frac{\partial J}{\partial \boldsymbol{o}^{(t)}} \frac{\partial \boldsymbol{o}^{(t)}}{\partial V} \\
&= \sum_t \frac{\partial J}{\partial \boldsymbol{o}^{(t)}} \boldsymbol{h}^{(t)} \\
\\
\frac{\partial J}{\partial \boldsymbol{c}} &= \sum_t \frac{\partial J}{\partial \boldsymbol{o}^{(t)}} \frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}} \\
&= \sum_t \frac{\partial J}{\partial \boldsymbol{o}^{(t)}} \\
\\
\frac{\partial J}{\partial \boldsymbol{h}^{(t)}} &= \frac{\partial J}{\partial \boldsymbol{h}^{(t+1)}} \frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}} + \frac{\partial J}{\partial \boldsymbol{o}^{(t)}} \frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \\
&= \frac{\partial J}{\partial \boldsymbol{h}^{(t+1)}} \frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{a}^{(t+1)}} \frac{\partial \boldsymbol{a}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}} + \frac{\partial J}{\partial \boldsymbol{o}^{(t)}} \frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \\
&= \frac{\partial J}{\partial \boldsymbol{h}^{(t+1)}} diag\left(\frac{\partial \tanh(\boldsymbol{a}^{(t+1)})}{\partial \boldsymbol{a}^{(t+1)}}\right) W + \frac{\partial J}{\partial \boldsymbol{o}^{(t)}} V
\end{aligned} \tag{11}$$

$$
\begin{aligned}
\frac{\partial J}{\partial \boldsymbol{a}^{(t)}} &= \frac{\partial J}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{a}^{(t)}} \\
&= \frac{\partial J}{\partial \boldsymbol{h}^{(t)}} diag\left(\frac{\partial \tanh(\boldsymbol{a}^{(t)})}{\partial \boldsymbol{a}^{(t)}}\right)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial J}{\partial W} &= \sum_t \frac{\partial J}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial W} \\
&= \sum_t \frac{\partial J}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{a}^{(t)}} \frac{\partial \boldsymbol{a}^{(t)}}{\partial W} \\
&= \sum_t \frac{\partial J}{\partial \boldsymbol{h}^{(t)}} diag\left(\frac{\partial \tanh(\boldsymbol{a}^{(t)})}{\partial \boldsymbol{a}^{(t)}}\right) \boldsymbol{h}^{(t-1)} \\
\frac{\partial J}{\partial U} &= \sum_t \frac{\partial J}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial U} \\
&= \sum_t \frac{\partial J}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{a}^{(t)}} \frac{\partial \boldsymbol{a}^{(t)}}{\partial U} \\
&= \sum_t \frac{\partial J}{\partial \boldsymbol{h}^{(t)}} diag\left(\frac{\partial \tanh(\boldsymbol{a}^{(t)})}{\partial \boldsymbol{a}^{(t)}}\right) \boldsymbol{x}^{(t)} \\
\frac{\partial J}{\partial \boldsymbol{b}} &= \sum_t \frac{\partial J}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}} \\
&= \sum_t \frac{\partial J}{\partial \boldsymbol{h}^{(t)}} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{a}^{(t)}} \frac{\partial \boldsymbol{a}^{(t)}}{\partial \boldsymbol{b}} \\
&= \sum_t \frac{\partial J}{\partial \boldsymbol{h}^{(t)}} diag\left(\frac{\partial \tanh(\boldsymbol{a}^{(t)})}{\partial \boldsymbol{a}^{(t)}}\right)
\end{aligned} \tag{12}
$$

5 LSTM

For the explanation of LSTM, please see colah's blog [4]. That is probably the best explanation available publicly. This section explains 'why' part as a generalization of *leaky units*[3][1].

The original problem with RNN is called *vanishing gradient* problem where the gradient vanishes or diverges during the backpropagation cycle depicted in figure 6. This is similar to repeated multiplication of square matrix, which diverges or converges along eigen vectors.

This is mitigated by leaky units as in figure 7. Pelase note that the recursion does not involve multiplication, but this is a kind of moving average. The leaky units use a fixed parameter α.

LSTM generalizes it as a learned parameter as in figure 8.

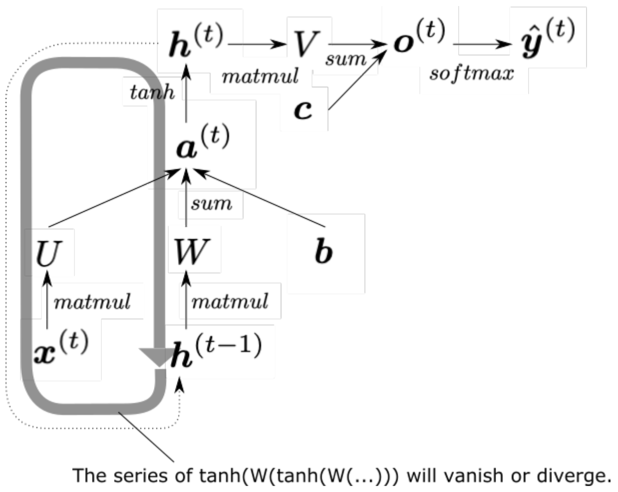


Figure 6: Gradient along the Back-propagation may Vanish or Diverge in RNN

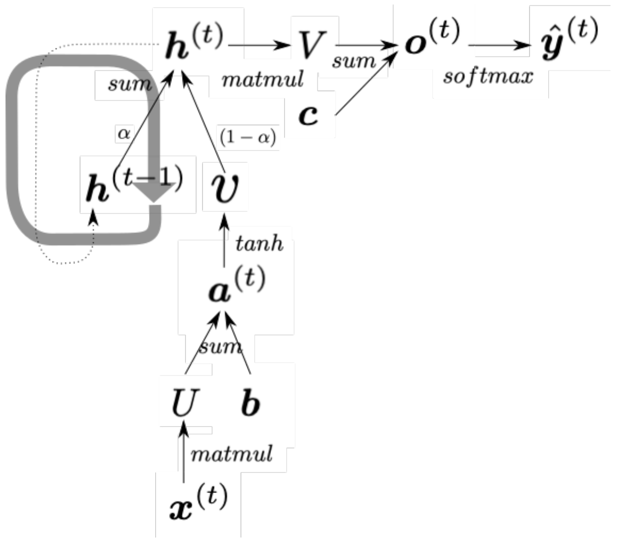


Figure 7: Recurrence with Leaky Units

The main part of LSTM is formulated as follows.

$$\boldsymbol{c}^{(t)} = \alpha_i \tanh\left(W_c \begin{bmatrix} \boldsymbol{h}^{(t-1)} \\ \boldsymbol{x}^{(t)} \end{bmatrix} + \boldsymbol{b}_c\right) + \alpha_f \boldsymbol{c}^{(t-1)} \tag{13}$$

$$\boldsymbol{h}^{(t)} = \alpha_o \tanh(\boldsymbol{c}^{(t)}) \tag{14}$$

The first term in equation 13 represents the traditional RNN part, and the second term is for the additive cell memory part, which is analogous to leaky

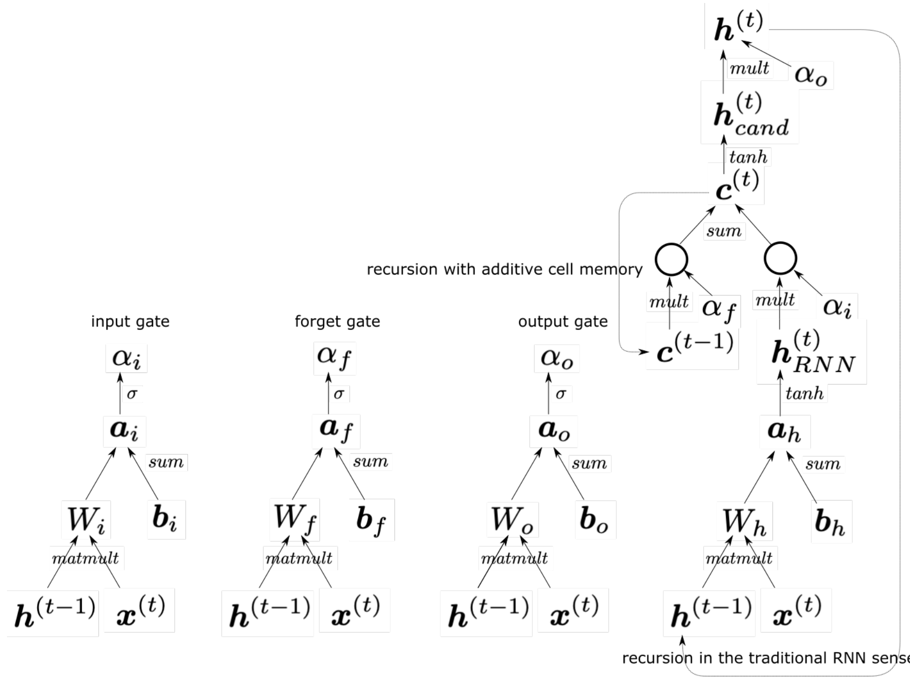


Figure 8: LSTM

unit. Those terms are controlled by the following gates. I think the tanh in equation 14 works as a layer-wise normalization.

$$
\begin{aligned}
\alpha_i &= \sigma\left(W_i \begin{bmatrix} \boldsymbol{h}^{(t-1)} \\ \boldsymbol{x}^{(t)} \end{bmatrix} + \boldsymbol{b}_i\right) \\
\alpha_f &= \sigma\left(W_f \begin{bmatrix} \boldsymbol{h}^{(t-1)} \\ \boldsymbol{x}^{(t)} \end{bmatrix} + \boldsymbol{b}_f\right) \\
\alpha_o &= \sigma\left(W_o \begin{bmatrix} \boldsymbol{h}^{(t-1)} \\ \boldsymbol{x}^{(t)} \end{bmatrix} + \boldsymbol{b}_o\right)
\end{aligned} \tag{15}
$$

6 Attention

Attention is a mechanism to handle variable length input and also variable length output for sequence-to-sequence learning. It is traditionally combined with bi-directional RNN. Plase see figure 9.

The blue parts indicate the attention mechanism. The lower part with $\boldsymbol{x}_i$, $\boldsymbol{h}_{f_i}$, and $\boldsymbol{h}_{b_i}$ is the standard bi-directional RNN of length Ni. We combine both the forward and the backward vector into one as follows.

$$
\boldsymbol{h}_i = \begin{bmatrix} \boldsymbol{h}_{f_i} \\ \boldsymbol{h}_{b_i} \end{bmatrix} \tag{16}
$$

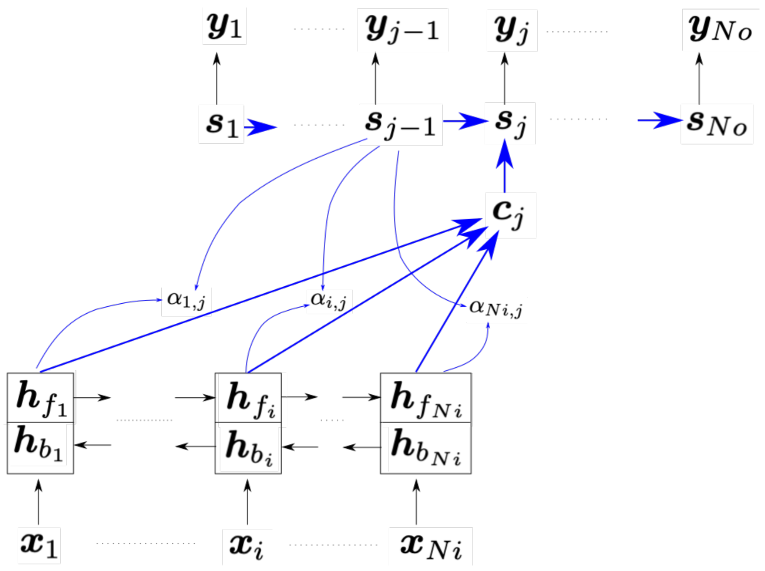


Figure 9: Attention Mechanism over Bi-Directional RNNs

The output $\boldsymbol{y}_j$ is generated from its corresponding hidden variable $\boldsymbol{s}_j$. $\boldsymbol{s}_j$ is generated from both $\boldsymbol{s}_{j-1}$ and a variable called context vector $\boldsymbol{c}_j$ An example is as follows, but this is not an essential part of the attention machanism. Any other generation will be possible.

$$\begin{aligned} \boldsymbol{y}_j &= W_y \boldsymbol{s}_j \\ \boldsymbol{s}_j &= sigmoid\left(W_s \begin{bmatrix} \boldsymbol{s}_{j_i} \\ \boldsymbol{c}_j \end{bmatrix}\right) \quad \text{element-wise} \end{aligned} \tag{17}$$

The context vector $\boldsymbol{c}_j$ is a weighted sum of the bi-directional hidden variables $\boldsymbol{h}_i$, i.e., attention into $\boldsymbol{h}_i$.

$$\boldsymbol{c}_j = \sum_{i}^{Ni} \alpha_{i,j} \boldsymbol{h}_i \tag{18}$$

The weights $\alpha_{i,j}$ are calculated from $\boldsymbol{s}_{j-1}$ and all the $\boldsymbol{h}_i$s as follows. First, we calculate a similarity between $\boldsymbol{s}_{j-1}$ and each $\boldsymbol{h}_i$. Usually one of the following is used.

$$\begin{aligned}
score_1\left(\boldsymbol{s}_{j-1}, \boldsymbol{h}_i\right) &= cosine\left(\boldsymbol{s}_{j-1}, \boldsymbol{h}_i\right) \\
score_2\left(\boldsymbol{s}_{j-1}, \boldsymbol{h}_i\right) &= \boldsymbol{v}_t^T \tanh\left(W_t \begin{bmatrix}\boldsymbol{s}_{j-1}\\ \boldsymbol{h}_i\end{bmatrix}\right) \\
score_3\left(\boldsymbol{s}_{j-1}, \boldsymbol{h}_i\right) &= \frac{1}{\sqrt{d}} \boldsymbol{s}_{j-1}^T \boldsymbol{h}_i \quad d \text{ is the length of } \boldsymbol{h}_i
\end{aligned} \tag{19}$$

Then weights $\alpha_{i,j}$ is calculated as follows.

$$\begin{bmatrix}\alpha_{1,j}\\ \alpha_{2,j}\\ \cdots\\ \alpha_{Ni,j}\end{bmatrix} = softmax\left(\begin{bmatrix} score_x\left(\boldsymbol{s}_{j-1}, \boldsymbol{h}_1\right)\\ score_x\left(\boldsymbol{s}_{j-1}, \boldsymbol{h}_2\right)\\ \cdots\\ score_x\left(\boldsymbol{s}_{j-1}, \boldsymbol{h}_{Ni}\right)\end{bmatrix}\right) \tag{20}$$

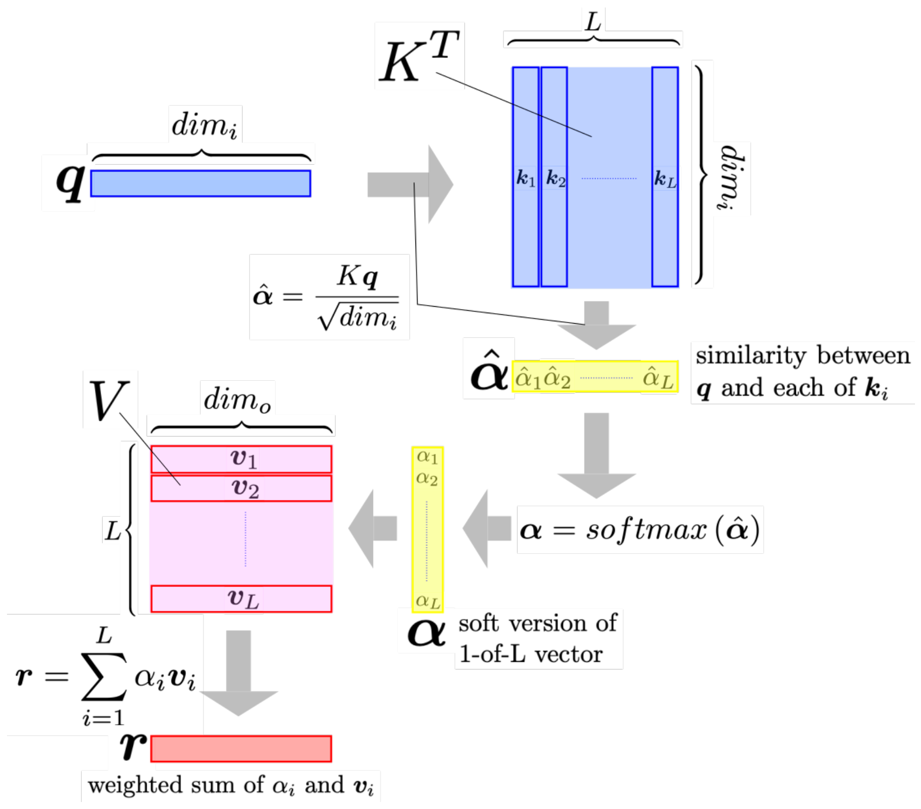
Here W_y, W_s, $\boldsymbol{v}_s$, and W_t are parameters to be learned.

7 Transformer

Transformer is an encoder-decoder mechanism to handle variable length input and output. This section explains Transformer by constructing it from the bottmom up to the top.

The bottom part is the attention mechanism, which works as a soft version of key-value query. The input to the query is a vector $\boldsymbol{q}$ of size dim_i. The output is a vector $\boldsymbol{r}$ of size dim_o. Assume the number of the key-value pairs is L. The key-value pairs are represented by two matrices $K \in \mathcal{R}^{L\times dim_i} = \begin{bmatrix}\boldsymbol{k}_1\\ \boldsymbol{k}_2\\ \cdots \boldsymbol{k}_{dim_i}\end{bmatrix}$ and $\in \mathcal{R}^{L\times dim_o} = \begin{bmatrix}\boldsymbol{v}_1\\ \boldsymbol{v}_2\\ \cdots \boldsymbol{v}_{dim_o}\end{bmatrix}$. For given $\boldsymbol{q}$ a suitable key $\boldsymbol{k}_i$ is chosen based on the similarity, and the corresponding $\boldsymbol{v}_i$ is returned as $\boldsymbol{r}$ This is *soft* in the sense that the key's similarity is measured by the scaled dot product, and the value is chosen by $softmax$. Please see figure 10

Next, we elaborate the single query into the multi-head attention used in Transformer. See figure 11. (a) corresponds to figure 10. (b) expands the single query to a sequence Q of length N_{input}. (c) introduces linear transformations W_Q, W_K, and W_V to Q, K, and V respectively. It usually reduces dimensionality. (d) represents the multi-head attention in which there are 8 parallel queries, each with its own linear transformations. The results are simply concatenated element-wise. For Transformer, the input is a sequence of a word embedding with relative positional embedding such as $cos(\theta_{word\_pos})$, $sin(\theta_{word\_pos})$. Above concatenated R sits leyer-wise normalization, element-wise fully connected NN, and layer-wise normalization with some skip connections, which can be seen in the original paper [5].



K^T
L
dim_i
$\boldsymbol{q}$
$\boldsymbol{k}_1$ $\boldsymbol{k}_2$ $\boldsymbol{k}_L$
dim_i
$\hat{\boldsymbol{\alpha}} = \frac{K\boldsymbol{q}}{\sqrt{dim_i}}$
$\hat{\boldsymbol{\alpha}}$ $\hat{\alpha}_1 \hat{\alpha}_2$ $\hat{\alpha}_L$
similarity between $\boldsymbol{q}$ and each of $\boldsymbol{k}_i$
V
dim_o
L
$\boldsymbol{v}_1$ $\boldsymbol{v}_2$ $\boldsymbol{v}_L$
α_1 α_2 α_L
$\boldsymbol{\alpha} = softmax\left(\hat{\boldsymbol{\alpha}}\right)$
$\boldsymbol{\alpha}$ soft version of 1-of-L vector
$\boldsymbol{r} = \sum_{i=1}^{L} \alpha_i \boldsymbol{v}_i$
$\boldsymbol{r}$
weighted sum of α_i and $\boldsymbol{v}_i$

Figure 10: Single Soft Query for K-V Pairs

References

[1] Salah El Hihi and Yoshua Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS95, page 493499, Cambridge, MA, USA, 1995. MIT Press.

[2] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[3] Michael C. Mozer. Induction of multiscale temporal structure. In *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS91, page 275282, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[4] Christopher Olah. Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 2020-03-30.

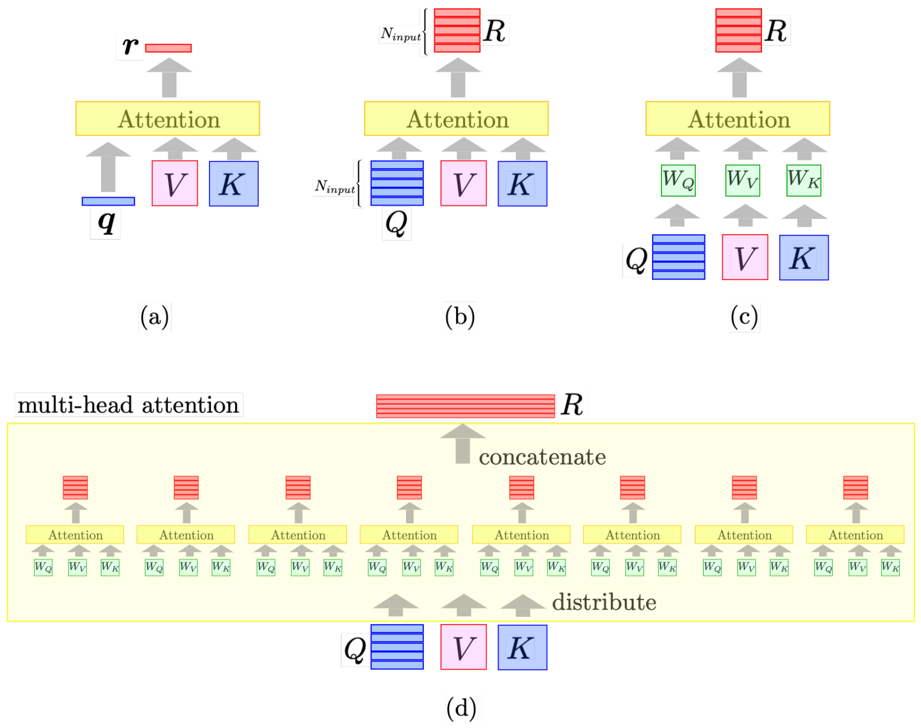


Figure 11: From a Single Input to Multi-Head Attention

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, undefinedukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS17, page 60006010, Red Hook, NY, USA, 2017. Curran Associates Inc.