Mathematics in Modern Natural Language Modeling

Richard Xu

February 27, 2023

1 A few words

To start this topic, I will first discuss the mathematics in modern natural language processing. By the way, Natural Language Processing (NLP) is one of the most important/exciting applications of artificial intelligence, machine learning and data mining. Contrary to computer vision, NLP will influence the work of many people in the future.

Although neural networks play an important role in NLP. However, we will cover neural networks later in this topic. Therefore, we temporarily avoid talking about N-N.

Also note that while this topic is about techniques in NLP, these techniques can also be applied to other machine learning and data science settings.

2 word embedding

Words are symbols: one may **not** able to perform arithmetic operations on them. They are suppose to be norminal attributes. (this is something I have discussed in the data mining subject)

However, in many NLP tasks, we need to operate on them as if they are vectors. So we must turn each word into a **vector**, e.g., the word "machine" $\rightarrow$ [2.4, 1.2, 1.9 . . .]

Once we do so, we can measure how similar or dissimilar between them. We can even perform "arithmetic" on them. A classic example from the the original word2vec paper would be:

$$\text{vec(King)} - \text{vec(man)} + \text{vec(woman)} = \text{vec(Queen)} \tag{1}$$

Mikolov et. al., (2013) "Linguistic Regularities in Continuous Space Word Representations" [1]

2.1 first attempt: one-hot encoding/embedding

a very naive and simple approach is to just use one-hot embedding, after all, students of MATH3836 (subject at last semester) should be very familiar with one-hot vector by now:

$$\begin{bmatrix} \text{“}a\text{”} & 1 & 0 & \cdots & 0 \\ \text{“}abbreviate\text{”} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \text{“}zoology\text{”} & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2}$$

however, the structure is huge and sparse. Think about the length of those vectors, they are as long as the total number of words in the vocabulary. In English, there is estimated of $40,000$ words.

In addition to the storage waste, one other disadvantage is that every pair of words are exactly $\sqrt{2}$ apart. So you can't really measure the similarity or dissimilarity between them.

The question is, can we do better?

2.2 second attempt: word2vec algorithm

The only data required are paragraphs of words, i.e., no extra human labeling is required. Unlike LDA models, with word-embeddings, we don't even need to know which word appear in which document.

2.2.1 conditional density using (target $\leftrightarrow$ context)

Although the paragraphs of words are the only information for the word2vec algorithm, however, there is still implicit labeling, i.e., we have the position of each word in the document. Therefore, we will use this information alone to construct the learning task:

1. Word2Vec algorithm leverage (*"target", "context"*) relationships to maximize the conditional probabilities.
2. which way should the condition be? the answer is both. Therefore it offers two approaches, i.e., to maximize one of the two conditional densities:
   (a) skip-gram: $\Pr(\text{"context"}|\text{"target"})$
   (b) continuous bag of words (CBOW): $\Pr(\text{"target"}|\text{"context"})$

So we need a methodology to construct context and target:

1. pick window size (odd number)
2. extract all tokens based on this chosen window size
3. remove middle word in each window; this becomes your **target** word, rest are **context**

An important question to consider is how to define this probability? Taking Skip-gram as an example (for CBOW, you can calculate it similarly), it is $\Pr(\text{"context" = "a specific word"}|\text{"target" })$, so this discrete probability should be defined on every word in the vocabulary. So it should return a probability vector as large as the vocabulary size!

What function should we consider to construct such a probability vector? you guessed right! Since we need a discrete probability, we can use the Softmax function. Let us generically define these probabilities in terms of the "center" (c) and the "output" word (o):

$$\frac{\exp(\mathbf{u}_o^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)} \tag{3}$$

and the entire softmax vectors can be represented as:

$$\left(\frac{\exp(\mathbf{u}_1^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)} \, , \, \cdots \, , \, \frac{\exp(\mathbf{u}_{o^\star}^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)} \, , \, \cdots \, , \, \frac{\exp(\mathbf{u}_{|\mathcal{V}|}^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)} \right) \tag{4}$$

and the index $o^\star$ is the one we are interested in.

2.2.2 skip-gram example: building training set

let's try **Skip-gram** of a window size 3, from the following sentence:

"the cat sit on the mat"

1. firstly, we generate all possible windows:

   (a) "the", "cat", "sit", target: cat

   (b) "cat", "sit", "on" , target: sit

   (c) "sit", "on ", "the", target: on

   (d) "on" , "the", "mat", target: the

   as far as this subject is concerned, we assume that for each target word, there are all context words on both sides. Therefore the word "the" will not be a context word

2. from each of the windows, the algorithm generate the input and output pairs for maximizing the conditional probabilities

$$(\mathbf{x}, y) = \text{(target, context)} \tag{5}$$

therefore we obtained:

$$\begin{aligned}
\text{"the"} &\leftarrow \text{"cat"} \\
\text{"sit"} &\leftarrow \text{"cat"} \\
\text{"cat"} &\leftarrow \text{"sit"} \\
\text{"on"} &\leftarrow \text{"sit"} \\
\text{"sit"} &\leftarrow \text{"on"} \\
\text{"the"} &\leftarrow \text{"on"} \\
\text{"on"} &\leftarrow \text{"the"} \\
\text{"mat"} &\leftarrow \text{"the"}
\end{aligned} \tag{6}$$

3. Using some initialization values, it is possible to take each context, target pair and maximize their conditional probability. Note that we need to maximize a number of conditions, each word has a chance to be a context and target word:

$$\begin{aligned}
&\Pr\left(\text{"the"}|\text{"cat"}\right) \times \Pr\left(\text{"sit"}|\text{"cat"}\right) \times \Pr\left(\text{"cat"}|\text{"sit"}\right) \times \Pr\left(\text{"on"}|\text{"sit"}\right) \times \\
&\Pr\left(\text{"sit"}|\text{"on"}\right) \times \Pr\left(\text{"the"}|\text{"on"}\right) \times \Pr\left(\text{"on"}|\text{"the"}\right) \times \Pr\left(\text{"mat"}|\text{"the"}\right)
\end{aligned} \tag{7}$$

we can express the above mathematically as, when choosing a window size $2m + 1$:

$$\mathcal{L} \equiv \prod_{n=1}^{N} \prod_{-m \le j \le m, j \neq 0} \Pr\left(w_{n+j} | w_n\right)$$
$$\log(\mathcal{L}) = \sum_{n=1}^{N} \sum_{-m \le j \le m, j \neq 0} \log\Big(\Pr\left(w_{n+j} | w_n\right)\Big) \tag{8}$$

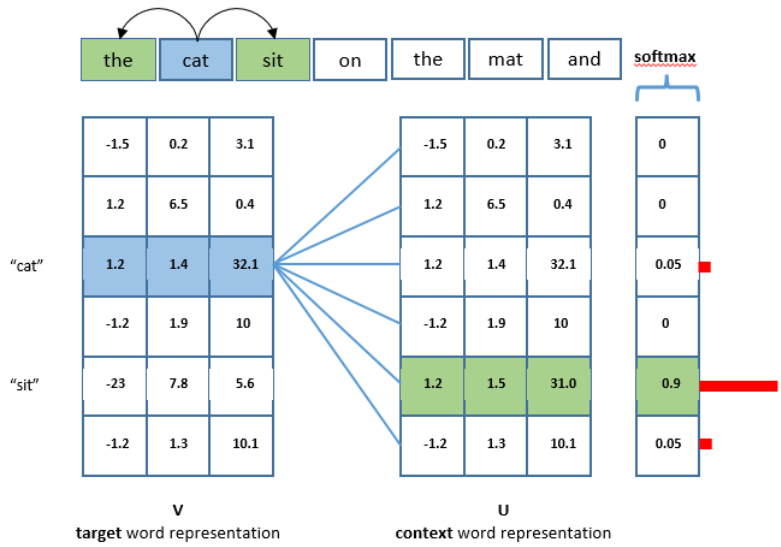
the outer sum tells us that each word n will get to be the "target" word once. The inner product is all the conditional densities using that target word.

4. For each word w_n, it has 2 representations $\mathbf{u}_n$ and $\mathbf{v}_n$, one for output (o) and one for context (c), and of course their size is much smaller than using one-hot!

For example, looking at a particular (i.e, the 2nd term in the product in Eq.(7):

$$\Pr\left(o = \text{"sit"} \middle| \, c = \text{"cat"}\right) \tag{9}$$

we need to maximize the conditional density of $o =$ context word given a $c =$ target word. We illustrate the above using the diagram below:



2.2.3 optimizing Skip-Gram objective function

we need to maximize the probability of $c =$ context word given a $t =$ target word.

In order to compute:

$$\arg\max_{\{\mathbf{v}\},\{\mathbf{u}\}} \left\{ \mathcal{L} \equiv \sum_{n=1}^{N} \sum_{-m \le j \le m, j \neq 0} \log\Big(\Pr\left(w_{n+j} | w_n\right)\Big) \right\} \tag{10}$$

We need to calculate $\frac{\partial \mathcal{L}}{\partial \mathbf{v}_t}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{u}_c}$, $\forall \mathbf{v}_t, \mathbf{u}_c \in \mathcal{V}$. However, due to symmetry, we only look at $\arg\max_{\mathbf{v}_t}$. Also, let's look at the derivative of a single term in the sum, where we usually write:

$$\Pr\left(w_{n+j}|w_n\right) \equiv \Pr(o|c) \equiv \Pr(\mathbf{u}_o|\mathbf{v}_c) \tag{11}$$

we can just perform sum of all the derivatives together later:

$$\begin{aligned}
\log(\Pr(o|c)) &= \log\left(\frac{\exp(\mathbf{u}_o^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}\right) \\
\frac{\partial \log(\Pr(o|c))}{\partial \mathbf{v}_c} &= \frac{\partial \mathbf{u}_o^\top \mathbf{v}_c}{\partial \mathbf{v}_c} - \frac{\partial \log\left(\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)\right)}{\partial \mathbf{v}_c} \\
&= \mathbf{u}_o - \left(\frac{\partial}{\partial \mathbf{v}_c} \log\Big(\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)\Big)\right) \\
&= \mathbf{u}_o - \left(\frac{1}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)} \frac{\partial}{\partial \mathbf{v}_c}\Big(\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)\Big)\right) \\
&= \mathbf{u}_o - \left(\frac{1}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}\Big(\sum_{o' \in \mathcal{V}} \frac{\partial}{\partial \mathbf{v}_c} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)\Big)\right) \\
&= \mathbf{u}_o - \frac{1}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}\Big(\sum_{o' \in \mathcal{V}} \mathbf{u}_{o'} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)\Big) \\
&= \mathbf{u}_o - \frac{\sum_{o' \in \mathcal{V}} \mathbf{u}_{o'} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)} \\
&= \mathbf{u}_o - \sum_{o' \in \mathcal{V}} \frac{\exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)}{\sum_{o'' \in \mathcal{V}} \exp(\mathbf{u}_{o''}^\top \mathbf{v}_c)} \mathbf{u}_{o'} \\
&= \mathbf{u}_o - \sum_{o' \in \mathcal{V}} \Pr(o'|c) \mathbf{u}_{o'} \\
&= \mathbf{u}_o - \mathbb{E}_{o' \sim \Pr(o'|c)}[\mathbf{u}_{o'}]
\end{aligned} \tag{12}$$

Obviously, when $|\mathcal{V}|$ is too large, the computational cost of computing the sum can be too high, there are many NLP mechanisms that can help us reduce the amount of computation. We will discuss them in the subsequent sections.

2.3 Simple CBoW example

very similar to to predict target word given multiple context words, where $\mathbf{u}_c$ become the average of context vectors.

3 Negative sampling

Let's use Skip-Gram model to demonstrate negative sampling. It is optimizing **different objective**, let $\theta = [\mathbf{u}, \mathbf{v}]$.

We let $\bar{w}$ to denote **negative samples**, meaning data coming from a negative population $\bar{D}$. In NLP, there could be many example of negative samples. For example, if one wishes to model skip-gram alike model, where words appear in the order of the text. Negative samples may mean words of any random ordering.

$$
\begin{aligned}
\theta &= \arg\max_{\theta} \prod_{(w,c)\in D} \Pr(D=1|w,c,\theta) \prod_{(\bar{w},c)\in \bar{D}} \Pr(D=0|\bar{w},c,\theta) \\
&= \arg\max_{\theta} \prod_{(w,c)\in D} \Pr(D=1|w,c,\theta) \prod_{(\bar{w},c)\in \bar{D}} (1-\Pr(D=1|\bar{w},c,\theta)) \\
&= \arg\max_{\theta} \sum_{(w,c)\in D} \log\left(\Pr(D=1|w,c,\theta)\right) + \sum_{(\bar{w},c)\in \bar{D}} \log\left(1-\Pr(D=1|\bar{w},c,\theta)\right) \\
&= \arg\max_{\theta} \sum_{(w,c)\in D} \log \frac{1}{1+\exp\left[-\mathbf{u}_w^\top \mathbf{v}_c\right]} + \sum_{(\bar{w},c)\in \bar{D}} \log\left(1-\frac{1}{1+\exp\left[-\mathbf{u}_{\bar{w}}^\top \mathbf{v}_c\right]}\right) \qquad \log(\cdot) \quad \text{is monotone} \\
&= \arg\max_{\theta} \sum_{(w,c)\in D} \log \sigma(-\mathbf{u}_w^\top \mathbf{v}_c) + \sum_{(\bar{w},c)\in \bar{D}} \log\left(\frac{1}{1+\exp\left[\mathbf{u}_{\bar{w}}^\top \mathbf{v}_c\right]}\right) \\
&= \arg\max_{\theta} \sum_{(w,c)\in D} \log \sigma(\mathbf{u}_w^\top \mathbf{v}_c) + \sum_{(\bar{w},c)\in \bar{D}} \log \sigma(-\mathbf{u}_{\bar{w}}^\top \mathbf{v}_c) \\
&= \arg\max_{\theta} \sum_{(w,c)\in D} \log \sigma(\mathbf{u}_w^\top \mathbf{v}_c) + \sum_{(\bar{w},c)\in \bar{D}} \log\left(1-\sigma(\mathbf{u}_{\bar{w}}^\top \mathbf{v}_c)\right)
\end{aligned}
\tag{13}
$$

for the last line we apply the property:

$$\sigma(-t) = 1-\sigma(t) \tag{14}$$

negative sampling based on Skip-Gram model, it is optimizing **different objective**, let $\theta = [\mathbf{u}, \mathbf{v}]$:

$$\theta = \arg\max_{\theta} \sum_{(w,c)\in D} \log \sigma(\mathbf{u}_w^\top \mathbf{v}_c) + \sum_{(\bar{w},c)\in \bar{D}} \log \sigma(-\mathbf{u}_{\bar{w}}^\top \mathbf{v}_c) \tag{15}$$

it still has a huge sum term $\sum_{(\bar{w},c)\in \bar{D}}(.)$, so we change to:

$$\theta = \arg\max_{\theta} \log \sigma(\mathbf{u}_w^\top \mathbf{v}_c) + \sum_{\bar{w}=1}^{k} \mathbb{E}_{\bar{w}\sim P(w)} \log \sigma(-\mathbf{u}_{\bar{w}}^\top \mathbf{v}_c) \tag{16}$$

sample a fraction of negative samples in second terms: $\{\bar{w}\}$ instead of going for $\forall(\bar{w} \neq w) \in \mathcal{V}$. We will discuss more about negative sampling when we talk about generative models.

4 Other interesting word representation

4.1 what is FastText?

Becamse popular in 2016, a library created by Facebook (now called Meta!) research team for efficient learning of word representations by Enriching Word Vectors with Subword Information.

for example, $n = 3$, i.e., 3-grams:

- **word**: *"where"*,
- **sub-words**: *"wh"*, *"whe"*, *"her"*, *"ere"*, *"re"*

So how is it different from Word2Vec? Instead of words, we now have **ngrams** of subwords, what is its **advantage**?

1. Helpful for finding representations for rare words
2. Give vector representations for words not present in dictionary

4.1.1 computation involving sub-words

We then represent a word $\mathbf{w}$ by the **sum of the vector representations** of all its n-grams. Concretely, in order to compute an un-normalised score $u(\cdot, \cdot)$ between $\mathbf{w}$ with center word $\mathbf{v}_c$:

Given a word $\mathbf{w}$ (e.g., "where"), $ns(\mathbf{w})$ is the set of n-grams appearing in $\mathbf{w}$, (e.g., *"wh"*, *"whe"*, *"her"*, *"ere"*, *"re"*), and $\{ \mathbf{z}_s \}$ is the representation to each individual n-gram $s \in ns(w)$ where its representation is simply the sum:

$$\mathbf{w} \equiv \sum_{s \in w} \mathbf{z}_s \tag{17}$$

therefore, now that the un-normalized inner product between center word $\mathbf{v}_c$ and $\mathbf{w}$ is:

$$u(\mathbf{w}, \mathbf{v}_c) = \exp\left[\sum_{s \in ns(w)} \mathbf{z}_s^\top \mathbf{v}_c \right] \tag{18}$$

4.2 Global Vectors for Word Representation(GloVe)

using the fact that **co-occurrence probabilities** of words are useful, GloVe learns word vectors through **word co-occurrences**.

I uses a co-occurrence matrix P where each entry P_{ij} describes how often word i appears in the context of word j. It also allows fast training and scalable to huge corpora. The objective function is:

$$\theta^* = \arg\min_\theta \left(J(\theta) \equiv \frac{1}{2} \sum_{\mathbf{u}_i \mathbf{v}_j \in \mathcal{V}} f(P_{ij})(\mathbf{u}_i^\top \mathbf{v}_j - \log P_{ij})^2 \right) \tag{19}$$

it tries to minimize difference:

$$\left(\mathbf{u}_i^\top \mathbf{v}_j - \log P_{ij}\right) \tag{20}$$

therefore, the more frequently two words appear together,more similar their vector representation should be, in terms of their inner product. $f(.)$ is weighting function to "prevent" certain scenarios, for example:

$$P_{ij} = 0 \implies \log P_{ij} = -\infty \implies f(0) = 0 \tag{21}$$

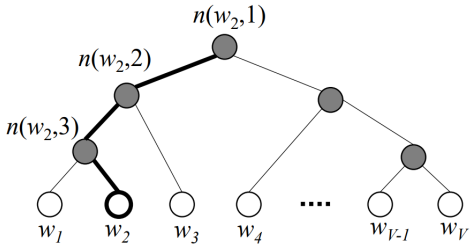
4.3 Hierarchical Softmax

This work was proposed in [2]

for word2vec algorithm, we compute explicitly the word representation of $\mathbf{u}_o$ as before using the softmax function, i.e., Eq.(4):

$$\left(\frac{\exp(\mathbf{u}_1^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)} , \cdots , \frac{\exp(\mathbf{u}_{o^\star}^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)} , \cdots , \frac{\exp(\mathbf{u}_{|\mathcal{V}|}^\top \mathbf{v}_c)}{\sum_{o' \in \mathcal{V}} \exp(\mathbf{u}_{o'}^\top \mathbf{v}_c)} \right)$$

However, the normalization part is computationally expensive. So the *idea* is, can we design a word2vec alike model that encapsulating all the information for every single word in the corpus, and then we can compute probabilities without the huge normalization term? Well, one way of doing so is is to represent the entire $\Pr(\mathbf{w}|c) \equiv \Pr(\mathbf{w}|\mathbf{u}_c)$ by:



Xin Rong, word2vec Parameter Learning Explained - it shows the corresponding path of $\mathbf{w}_2$

Note that $\mathbf{w}$ is fixed and $\mathbf{u}_c$ is the input variable. **super advantage**: $\Pr(\mathbf{w}|\mathbf{v}_c)$ is already a probability by multiplying all probabilities of path, no need to normalize!

4.3.1 definition

1. Each word in this softmax tree, i.e., $\mathbf{w}_i$ has a unique (pre-defined) path, which performs a left or right turn from nodes: $n(w_i, 1), n(w_i, 2), n(w_i, 3), \ldots$ note $n(\cdot)$. these are purely symbolic.

2. the route is defined in such a way that, each child node is from a (LEFT/RIGHT) "channel" of its parent: i.e., $n(w, j+1) = \text{ch}(n(w, j))$, for example:

$$\begin{aligned} n(w_2, 2) &= \text{LEFT}\ (n(w_2, 1)) \\ n(w_2, 3) &= \text{LEFT}\ (n(w_2, 2)) \\ \underbrace{n(w_2, 4)}_{w_2} &= \text{RIGHT}(n(w_2, 3)) \end{aligned} \tag{22}$$

3. there are V words in leaf (white node), there are $V-1$ inner (non-leaf) nodes (grey node)

4. each node associates with a vector $\mathbf{v}'$ which is shared among all words going through this node. note that we use $\mathbf{v}'_n$ instead of $\mathbf{v}_n$ to indicate that this is not word representation

4.3.2 compute the probabilities

$$\text{we define:} \quad \text{branch}[.] = \begin{cases} 1: & \text{LEFT} \\ -1: & \text{RIGHT} \end{cases} \tag{23}$$

now that we substitute a context $\mathbf{u}_c$ (skip-gram or CBOW), we can obtain its probability as:

$$\Pr(w|c) = \prod_{j=1}^{L(w)-1} \sigma\Big(\underbrace{\text{branch}\big[n(w,j+1) = \text{ch}(n(w,j))\big]}_{\text{the sign}} \mathbf{v}_{n(w,j)}^{\top} \mathbf{u}_c \Big) \tag{24}$$

note that $\text{branch}\big[n(w,j+1) = \text{ch}(n(w,j))\big]$ is given, i.e., this is determined form the configuration of the tree itself. If one adds up all the probabilities of the leaf node, one can see that, for any input context $\mathbf{u}_c$:

$$\sum_{\mathbf{w} \in \mathcal{V}} p(\mathbf{w}|\mathbf{u}_c) = 1 \tag{25}$$

4.3.3 example of $\Pr(\mathbf{w}_2|\mathbf{u}_c)$ and $\Pr(\mathbf{w}_3|\mathbf{u}_c)$

- at root level, i.e., $n(\cdot, 1)$, $n(w_2, 1) = n(w_3, 1)$ in fact, the values in the set $\big\{n(w_i, 1)\big\}_{i=1}^{|\mathcal{V}|}$ all equal
- $n(w_2, 2) = n(w_3, 2)$ as both w_2 and w_3 share the same path from 1 to 2
  therefore, we can compute the probabilities as:

$$\begin{aligned} \Pr(w_2|c) &= p\big(n(w_2,1), \text{LEFT}\big) p\big(n(w_2,2), \text{LEFT}\big) p\big(n(w_2,3), \text{RIGHT}\big) \\ &= \sigma\Big({\mathbf{v}'}_{n(w_2,1)}^{\top} \mathbf{u}_c \Big) \sigma\Big({\mathbf{v}'}_{n(w_2,2)}^{\top} \mathbf{u}_c \Big) \sigma\Big(-{\mathbf{v}'}_{n(w_2,3)}^{\top} \mathbf{u}_c \Big) \\ \Pr(w_3|c) &= p\big(n(w_3,1), \text{LEFT}\big) p\big(n(w_3,2), \text{RIGHT}\big) \\ &= \sigma\Big({\mathbf{v}'}_{n(w_3,1)}^{\top} \mathbf{u}_c \Big) \sigma\Big(-{\mathbf{v}'}_{n(w_3,2)}^{\top} \mathbf{u}_c \Big) \end{aligned} \tag{26}$$

5 Overview about Natural Language Processing Tasks

5.1 major NLP tasks

As mentioned earlier, there are too many NLP tasks. Therefore, we list some areas of active research fields:

- **machine translation**: *encoder to decoder*

  automatically translate text from one human language to another, for example, English to Chinese. Since 2014, Neural Machine Translation (NMT) dominates!

- **text summerization**: *context to decoder*

  1. **Extraction-based summarization** extracts objects (part-sentences or words) form the long document without modification, i.e., pick the important bits

2. **abstraction-based summarization**
   involves paraphrasing sections of the source document

- **Q and A**: *encoder to decoder given context*

the above three (3) may share a design architecture/elements

5.2 down-stream tasks

Too many of applications, and I list a few example works done by my former students.

- **natural language generation** by learning document corpus
  generate natural language from a machine representation, or for machine to generate semantically-similar texts given a training corpus
- **chatbot**
  enable human and machine to communicate using natural language
- **natural language to cross-domain translation**
  1. NLP to image
  2. NLP to animation
- **topic modeling**
  this is **unsupervised learning**, tries to assign each document in the document corpus a latent distribution of topics

In the rest of the topics, let me describe some of the modern "classic" NLP tasks:

6 Beam search

Let's talk a little about natural language generation, or NLG. or In Deep NLP terms, the Decoder generates words **jointly**, so how we may compute:

$$\{\widehat{y}_1, \ldots, \widehat{y}_T\} = \underset{y_1,\ldots,y_T}{\arg\max} \left[\Pr(y_1, \ldots, y_T|\mathbf{x}) \equiv \Pr(y_1|\mathbf{x}) \Pr(y_2|y_1, \mathbf{x}), \ldots, \Pr(y_T|y_1, \ldots, y_{T-1}, \mathbf{x}) \right] \quad (27)$$

but as the depth of the tree, i.e., T became large, compute probability for each and every single chain became unmanageable. Let's look at the following two extreme scenarios:

1. **select all**: tree-width $= N$, so we get answer to be:

$$\{\widehat{y}_1, \ldots, \widehat{y}_T\} = \underset{y_1,\ldots,y_T}{\arg\max} \left[\Pr(y_1|\mathbf{x}) \Pr(y_2|y_1, \mathbf{x}) \Pr(y_3|y_1, y_2, \mathbf{x}) \ldots, \Pr(y_T|y_1, \ldots, y_{T-1}, \mathbf{x}) \right] \quad (28)$$

in each depth, keep (select) **full width** N, until its full depth T, before select a best path (accurate, but computationally infeasible): N^T paths!

2. **select one**: tree-width $= 1$, **greedy algorithm** (*def. making locally optimal choice at each stage, to "approximately" a global optimum*)

   select best word at each depth t: choose one branch in a depth, and discard rest of sibling branches (fast & storage efficient, but accuracy-wise bad)

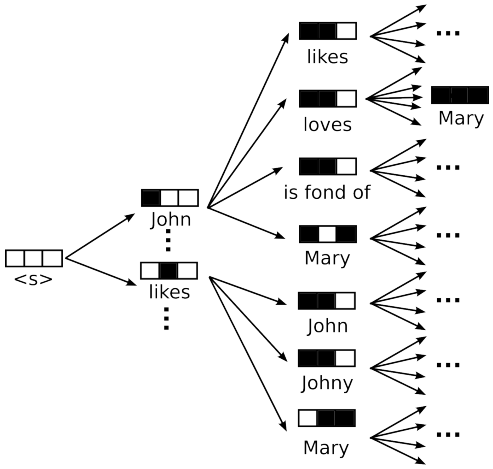


image credit: internet

$$\{\widehat{y}_1, \ldots, \widehat{y}_T\} \approx \{\tilde{y}_1, \ldots, \tilde{y}_T\}$$
$$= \left\{ \tilde{y}_1 \equiv \arg\max_{y_1} \Pr(y_1|\mathbf{x}),\ \tilde{y}_2 = \arg\max_{y_2} \Pr(y_2|\tilde{y}_1, \mathbf{x}), \ldots, \tilde{y}_T \equiv \arg\max_{y_T} \Pr(y_T|red\tilde{y}_1, \ldots, \tilde{y}_{T-1}, \mathbf{x}) \right\} \tag{29}$$

6.1 trade off?

Obviously, having tree width of N and 1 are both **not ideal**, so we go for a comprise:
easy , why don't we select tree width W, such that

$$1 < W < N \tag{30}$$

loop from 1 to T, at each depth t:

1. use W **most probable** branches chosen at the previous depth
2. extend each W branch by depth $+1$, and to generate $W \times N$ candidate branches
3. choose the W **most probable** branches, and go for next iteration

6.2 beam-search: normalization

1. **problem:** of beam-search is that the words are generated from:

$$\Pr(y_1, \ldots, y_T|\mathbf{x}) = \Pr(y_1|\mathbf{x}) \Pr(y_2|y_1, \mathbf{x}) \Pr(y_3|y_1, y_2, \mathbf{x}) \Pr(y_T|y_1, \ldots, y_{T-1}, \mathbf{x}) \tag{31}$$

   therefore, shorter the word, higher the probability (less to multiply)

2. **solution:** beam-search normalization

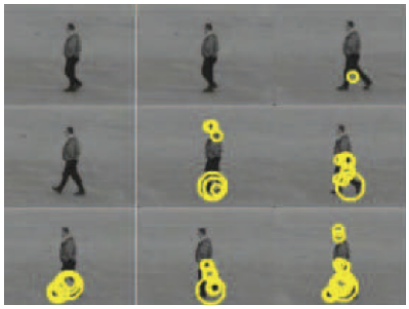
   one simple example: Andrew Ng's (2017) DL course:

$$\begin{aligned} \{\widehat{y}_1, \ldots, \widehat{y}_T\} &= \arg\max_{y_1,\ldots,y_T} \Pr(y_1, \ldots, y_T|\mathbf{x}) = \Pr(y_1|\mathbf{x}) \Pr(y_2|y_1, \mathbf{x}), \ldots, \Pr(y_T|y_1, \ldots, y_{T-1}, \mathbf{x}) \\ &= \arg\max_{y_1,\ldots,y_T} \left(\frac{1}{T^\alpha} \sum_{t=1}^{T} \log \Pr(y_t|y_1, \ldots, y_{t-1}, \mathbf{x}) \right) \end{aligned} \tag{32}$$

   the method is not new, it's called **geometry mean**:

$$\left(\prod_{i=1}^{T} p_i \right)^{\frac{1}{T}} = \exp \left[\frac{1}{T} \sum_{i=1}^{T} \log p_i \right] \tag{33}$$

I also used geometry mean to address problem of variable number of features at each image in a sequence



HMM-MIO: An enhanced hidden Markov model for action recognition [3]

6.3 more sophisticated normalization

Based on the paper, *Wu et. al., (2016), "Google's Neural Machine Translation System: Bridging the Gap"*. The aim is to **maximize** scores generated by the model:

$$s(Y, X) = \frac{1}{\text{lp}(Y)} \log P(Y|X) + \text{cp}(X, Y) \tag{34}$$

where X is the input sentence and the Y is the output sentence. We use $|Y|$ to indicate the length of Y. Notice that there are two penalties used, the first one is "length penalty" or lp, the second is "coverage penalty" or cp.

6.3.1 Length penalty lp(Y)

Since we try to maximize $s(Y, X)$. Therefore, you would like lp(Y) to be as small as possible. It is inversely proportional to $s(Y, X)$. In words, it tries encourage the length of output Y do not become too long.

the most obvious way of defining lp is lp$^* = |Y|^\alpha$, similar to the previous method, i.e., Eq.(32), where α can be used to tune rate of increase.

However, a different and more sophisticated normalization lp(Y) can be defined:

$$\text{lp}(Y) = \frac{(5 + |Y|)^\alpha}{(5 + 1)^\alpha} \tag{35}$$

Let's plot them both so we can see their difference:

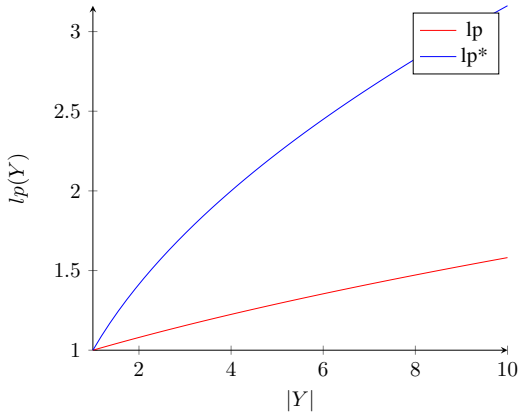


Figure 1: compare and constrast the two length panelties, lp$(Y) = \frac{(5+|Y|)^\alpha}{(5+1)^\alpha}$ and lp$^*(Y) = |Y|^\alpha$

When $|Y|$ is small, the difference between lp(Y) and lp$^*(Y)$ is not as significant. However, for lp(Y), the penalty increases "slowly" as $|Y|$ increases, while lp$^*(Y)$ increases much faster. In addition, the effect of having the value 5 in it (based on empirical studies presumably). This can be seen as:

6.3.2 coverage penalty

The second part of the penalty is the coverage penalty or cp, where the objective function is also maximized:

$$s(Y, X) = \frac{1}{\text{lp}(Y)} \log P(Y|X) + \text{cp}(X, Y) \tag{36}$$

it needs also to maximize **coverage penalty**:

$$\text{cp}(X, Y) = \beta \sum_{j=1}^{|X|} \log \Big(\min \big(\sum_{i=1}^{|Y|} a_{i,j}, 1.0 \big) \Big) \tag{37}$$

- where $a_{i,j}$ is attention probability of i-th target **decoder** word on j-th source **encoder** word (technically via their "states", but we use word in here). Sorry to bring this up in here before we formally talk about it in section (7). If we can portray attention as a matrix, then:

$$\mathbf{a} = \begin{bmatrix} a_{i=1,j=1} & a_{i=1,j=2} & a_{i=1,j=3} \\ a_{i=2,j=1} & a_{i=2,j=2} & a_{i=2,j=3} \\ a_{i=3,j=1} & a_{i=3,j=2} & a_{i=3,j=3} \end{bmatrix} \tag{38}$$

- we know that:

$$\sum_{j=1}^{|X|} a_{i,j} = 1 \quad \text{and} \quad \sum_{i=1}^{|Y|} a_{i,j} \neq 1 \text{ in general} \tag{39}$$

  i.e, each row of $\mathbf{a}$ needs to add-up to 1 but each column of $\mathbf{a}$ does **not** need to add-up to 1

- think $\mathbf{a}$ as a matrix of size $|Y| \times |X|$, which we distribute a total mass of $|Y|$ in value among all of its elements, for example, $|X| = |Y| = 3$:

$$\underbrace{\mathbf{a} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}}_{\textbf{minimized } \mathrm{cp}(X,Y)} \qquad \underbrace{\mathbf{a} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\textbf{maximized } \mathrm{cp}(X,Y)} \tag{40}$$

- favor translations that fully cover source sentence according to the attention module, i.e., there is one corresponding encoder word for each decoder word.. (jokingly, the reverse may correspond how I translate Cantonese to English/Mandarin at the moment)

6.4 More sophisticated normalization

- lastly, one may encourage the decoder to be longer than the encoder (which is fixed as we do know the input): `opennmt.net/OpenNMT/translation/beam_search/`. It is natural to assume that the longer the sentence from the input, the longer the sentence of the output.

$$\mathrm{ep}(X, Y) = \gamma \frac{|Y|}{|X|} \tag{41}$$

7 Sequence to Sequence with Attention

this work was proposed in [4] or [5]. The pure RNN-base seq2seq was described in [6].

- **encoders** have hidden states: $\{\mathbf{z}_1, \ldots, \mathbf{z}_m\} \in \mathbb{R}^d$. This is the latent representation of words feed into the encoder. (language to be translated from)

- **decoders** have hidden states: $\{\mathbf{h}_1, \ldots, \mathbf{h}_n\} \in \mathbb{R}^d$. This is the latent representation of words of the decoder. (language to be translated to)
- compute **dot-product**: $\mathbf{h}_i^\top \mathbf{z}_j$. this is how aligned (in the latent representation sense) between i^{th} state in the encoder and j^{th} state in the decoder.
- **attentions** between i^{th} decoder state and j^{th} encoder state is:

$$a_{ij} \equiv a_{i \leftarrow j} = \frac{\exp\left(\mathbf{h}_i^\top \mathbf{z}_j\right)}{\sum_{j'=1}^{m} \exp\left(\mathbf{h}_i^\top \mathbf{z}_{j'}\right)} \tag{42}$$

this is normalized (in terms of probability). Showing how much does i^{th} word in the decoder receives from j^{th} word in the encoder. In terms of translation, it means in translating i^{th} word, how much attention (what proportion) does it receive from the j^{th} word from the original language. Obviously:

$$\sum_{j=1}^{m} a_{ij} = 1 \tag{43}$$

- each i^{th} decoder has $\mathbf{a}_i = \{a_{i,1}, \ldots, a_{i,m}\}$ attention weights/probability vector of the encoder
- **condition vector** $\mathbf{c}_i$ for each decode word i:

$$\mathbf{c}_i = \sum_{j=1}^{m} a_{i,j} \mathbf{z}_j \tag{44}$$

it is a convex combination of encoder states each weighted by its contribution/attention to decoder state i. Therefore $\mathbf{c}_i$ contains the attention information.

- new **augmented decoder state** $\tilde{\mathbf{h}}_i$:

$$\tilde{\mathbf{h}}_t = [\mathbf{c}_i; \mathbf{h}_t] \in \mathbb{R}^{2d} \tag{45}$$

7.0.1 some new variations

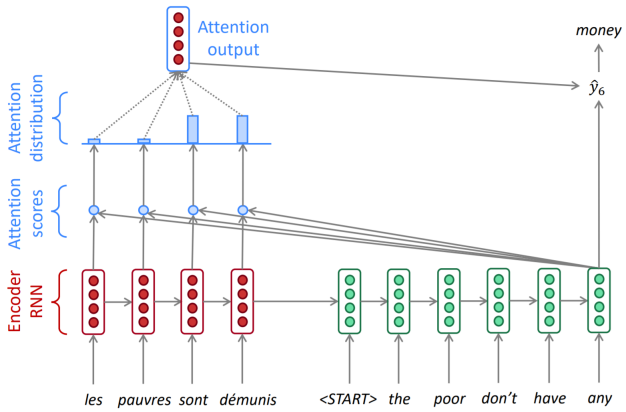


image from internet

In recent NLP literature, many version exist for **dot-product**: $\mathbf{h}_i^\top \mathbf{z}_j$

1. enable h and z have different dimensionality

$$\mathbf{h}_i^\top \mathbf{W} \mathbf{z}_j \tag{46}$$

2. alternatively:

$$\mathbf{v}_i^\top \tanh(\mathbf{W_1} \mathbf{h}_i + \mathbf{W_2} \mathbf{z}_j) \tag{47}$$

$(\mathbf{v}_i, \mathbf{W_1}, \mathbf{W_2})$ are parameters of this dot-product. Pointer Networks uses this!

7.1 Improvements

- **issue one**: decoder sometimes repeat themselves (e.g. "machine learning machine learning ..."). It happens when the attention probably vector for $\{\mathbf{h}_t\}$ are similar

  solution we will use [7]

  - **coverage vector** Sum of attention distributions so far, say up to word t, we have:

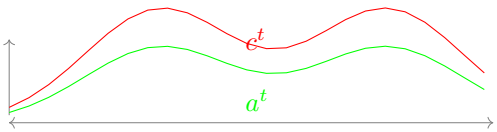
$$\mathbf{c}^t = \sum_{t'=0}^{t-1} \mathbf{a}^{t'} \tag{48}$$

  - penalize overlap between **coverage vector** $\mathbf{c}^t$ and new attention distribution $\mathbf{a}^t$:

$$\text{covloss}_t = \sum_i \min(\mathbf{a}_i^t, \mathbf{c}_i^t) \tag{49}$$

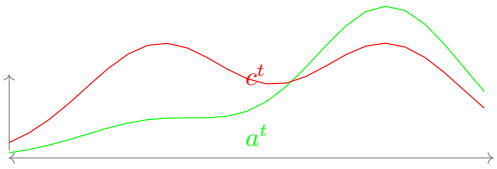
  - the above equation can be understood as follows. (although we are using a continuous representation):

    imagine $c_i^t \geq a_i^t \forall i$, then $\text{covloss}_t = 1$, which is its **maximum**. this happens when covloss_t is a multilicative envelop of a^t:



$\text{covloss}_t = \sum_i a_i^t = 1$ at maximum

this is the situation that you want to avoid, as the current attention is $\mathbf{a}^t$ has the same shape as all previous attentions $\mathbf{c}^t$. So it is more likely to generate repeating words.
alternatively, the following is desirable:



$\text{covloss}_t < 1$

- in essence, minimizing covloss_t makes $\mathbf{a}^t$ distributed differently to $\mathbf{c}^t$.
- not otherwise stated, however, I believe $\mathbf{c}^t$ may also be calculated as:

$$\mathbf{c}^t = \frac{1}{t}\sum_{t'=0}^{t-1} \mathbf{a}^{t'} \tag{50}$$

7.2 Improvement

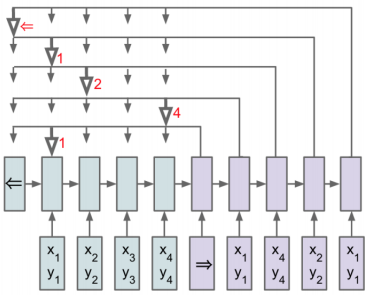
- **issue two** decoder may not able to translate "out-of-vocabulary words" such as names of a company

- suppose to have the following text summerization task:
  - **original text**:
    "The ShatinCo has made all reasonable efforts to ensure that this material has been reproduced with the consent of TaiwaiCo"
  - **summerized text**:
    "TaiwaiCo allowed ShatinCo to reuse its content"
  - some of the word should appear **as is it**
- RNN-based summarization may replace "Mary" with "Jane" and "Sydney" with "Melbourne" since these word embedding tend to cluster (and hence their dot product are similar!)
- **solution** "Pointer Networks" may be handy to comes to help!

7.2.1 What is Pointer Networks anyway?

In Pointer Networks [8] described as follows:

For pointer networks, the input is a given set of data points. Note that the raw input data has no sequence. they form purely a set. By designing a network stricture that looks like this:

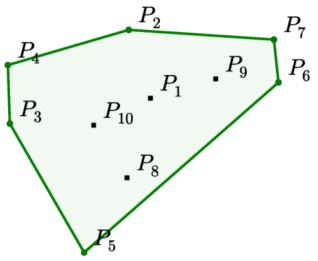


It will output a **series of positions** relative to the original data. In a way it reorders the original "arbitrary" input sequence (since the original data does not form any order). Now that the new order becomes useful.

So for training data, the "correct order" of data is its output y.

7.2.2 use for mathematics

it could solve combinatorial geometry problems:



we compute:

$$\Pr(C_i|C_1, \ldots, C_{i-1}, \mathcal{P}) \tag{51}$$

where $\mathcal{P} = \{P_1, \ldots\}$

Question for the students: how do you get the training data to solve the above problem? By the way, this is an example of how one can use neural networks to solve difficult mathematics problems.

7.2.3 back to NLP

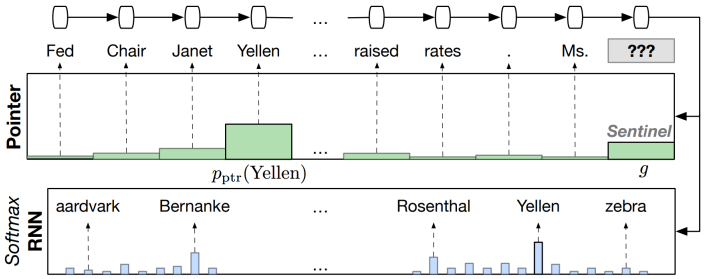
- "Seq2Seq with attention" is to predict **content** of next word
- "Pointer Networks" is to predict next **position** of encoding sequence
- uses a different "inner product " function: $v_i^\top \tanh(\mathbf{W_1} h_i + \mathbf{W_2} z_j)$

$$a_{ij} = \frac{\exp\left(\mathbf{v}_i^\top \tanh(\mathbf{W_1 h}_i + \mathbf{W_2 z}_j)\right)}{\sum_{j'=1}^{m} \exp\left(\mathbf{v}_i^\top \tanh(\mathbf{W_1 h}_i + \mathbf{W_2 z}_{j'})\right)} \tag{52}$$

- now that we apply Pointer Network to "copy" rare words from encoder to decoder, what about generating words that don't appear in the encoder?
- the answer is a **mixture model** that does both **copy (extraction)** and **generation (abstraction)**

7.2.4 Pointer Sentinel Mixture Models

- Pointer sentinel mixture models [9]
- combines **abstraction** and **extraction** together



$$p(\text{“}Yellen\text{”}) = g \times p_{\text{vocab}}(\text{“}Yellen\text{”}) + (1 - g) \times p_{\text{ptr}}(\text{“}Yellen\text{”}) \tag{53}$$

- g is mixture gate, uses sentinel to dictate how much probability mass to give to vocabulary
- note that PSMM paper doesn't discuss seq2seq, instead it is about generate $\Pr(y_N | w_1, \ldots, w_{N-1})$

8 Transformer Architecture

8.1 Dot-Product Attention

8.1.1 single query $\mathbf{q}$

given a single query vector $\mathbf{q}$, and matrix of keys $\mathbf{K}$ and values $\mathbf{V}$:

$$\mathbf{q} \equiv \underbrace{\begin{bmatrix} - & \mathbf{q} & - \end{bmatrix}}_{\in \mathbb{R}^{1 \times d_k}} \qquad \mathbf{K} \equiv \underbrace{\begin{bmatrix} - & \mathbf{k}_1 & - \\ - & \dots & - \\ - & \mathbf{k}_m & - \end{bmatrix}}_{\in \mathbb{R}^{m \times d_k}} \qquad \mathbf{V} \equiv \underbrace{\begin{bmatrix} - & \mathbf{v}_1 & - \\ - & \dots & - \\ - & \mathbf{v}_m & - \end{bmatrix}}_{\in \mathbb{R}^{m \times d_v}} \tag{54}$$

and let $(\mathbf{q}, \mathbf{K}, \mathbf{V})$ be tuples. Bear in mind that in Eq.(54), $\mathbf{q}$ is a row vector. This is to conform with Eq.(58), where all the $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ have each element expressed as a row vector.

the Dot-Product Attention (DPA) is defined as:

$$\begin{aligned}
\implies \mathbf{q}\mathbf{K}^\top &= \begin{bmatrix} \underbrace{\mathbf{q}\mathbf{k}_1^\top}_{\in \mathbb{R}} & \dots & \underbrace{\mathbf{q}\mathbf{k}_m^\top}_{\in \mathbb{R}} \end{bmatrix} \\
\implies A(\mathbf{q}, \mathbf{K}, \mathbf{V}) &\equiv \text{softmax}(\mathbf{q}\mathbf{K}^\top)\mathbf{V} \\
&= \text{softmax}\Big(\begin{bmatrix} \mathbf{q}\mathbf{k}_1^\top & \dots & \mathbf{q}\mathbf{k}_m^\top \end{bmatrix} \Big) \begin{bmatrix} - & \mathbf{v}_1 & - \\ - & \dots & - \\ - & \mathbf{v}_m & - \end{bmatrix} \\
&= \sum_{i=1}^{m} \underbrace{\frac{\exp[\mathbf{q}\mathbf{k}_i^\top]}{\sum_j \exp[\mathbf{q}\mathbf{k}_j^\top]}}_{\mathbf{a}_i} \mathbf{v}_i \\
&\in \mathbb{R}^{d_\mathbf{v}}
\end{aligned} \tag{55}$$

8.1.2 in the case of "seq2seq with attention"

we have:

$$\begin{aligned}
\mathbf{q} &\equiv \mathbf{h}_i \\
\mathbf{k}_i = \mathbf{v}_i &= \mathbf{z}_i \\
\implies A(\mathbf{q}, \mathbf{K}, \mathbf{V}) &\equiv A(\mathbf{h}_i, \mathbf{Z}, \mathbf{Z}) = c_i
\end{aligned} \tag{56}$$

where is our **conditional** or **context** vector. In order to confirm the notation in Eq.(54), we have assumed $\mathbf{h}_i$ to be a row vector, so instead of $\mathbf{h}_i^\top \mathbf{z}_j$, we express it as $\mathbf{h}_i \mathbf{z}_j^\top$:

$$a_{ij} = \frac{\exp\left(e_{i,j}\right)}{\sum_{t=1}^{m} \exp\left(e_{i,t}\right)} = \frac{\exp\left(\mathbf{h}_i \mathbf{z}_j^\top\right)}{\sum_{t=1}^{m} \exp\left(\mathbf{h}_i \mathbf{z}_t^\top\right)} \tag{57}$$

8.1.3 many queries

now we have many $\mathbf{Q} = \{\mathbf{q}_i\}$, e.g., N words in the decoder, we now have $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ expressed as:

$$\mathbf{Q} \equiv \underbrace{\begin{bmatrix} - & \mathbf{q}_1 & - \\ - & \dots & - \\ - & \mathbf{q}_n & - \end{bmatrix}}_{\in \mathbb{R}^{n \times d_k}} \qquad \mathbf{K} \equiv \underbrace{\begin{bmatrix} - & \mathbf{k}_1 & - \\ - & \dots & - \\ - & \mathbf{k}_m & - \end{bmatrix}}_{\in \mathbb{R}^{m \times d_k}} \qquad \mathbf{V} \equiv \underbrace{\begin{bmatrix} - & \mathbf{v}_1 & - \\ - & \dots & - \\ - & \mathbf{v}_m & - \end{bmatrix}}_{\in \mathbb{R}^{m \times d_v}} \tag{58}$$

$A(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\mathbf{Q}\mathbf{K}^\top)\mathbf{V}$ can be expressed as follows:

$$\begin{aligned} \implies \mathbf{Q}\mathbf{K}^\top &= \begin{bmatrix} \mathbf{q}_1\mathbf{k}_1^\top & \dots & \mathbf{q}_1\mathbf{k}_m^\top \\ \dots & \dots & \dots \\ \mathbf{q}_n\mathbf{k}_1^\top & \dots & \mathbf{q}_n\mathbf{k}_m^\top \end{bmatrix} \\ \implies A(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &\equiv \text{softmax}(\mathbf{Q}\mathbf{K}^\top)\mathbf{V} \\ &= \begin{bmatrix} \text{softmax}\big(\begin{bmatrix} \mathbf{q}_1\mathbf{k}_1^\top & \dots & \mathbf{q}_1\mathbf{k}_m^\top \end{bmatrix}\big) \\ \vdots \\ \text{softmax}\big(\begin{bmatrix} \mathbf{q}_n\mathbf{k}_1^\top & \dots & \mathbf{q}_n\mathbf{k}_m^\top \end{bmatrix}\big) \end{bmatrix} \begin{bmatrix} - & \mathbf{v}_1 & - \\ - & \dots & - \\ - & \mathbf{v}_m & - \end{bmatrix} \\ &= \underbrace{\begin{bmatrix} \underbrace{\sum_{i=1}^{m} \frac{\exp[\mathbf{q}_1\mathbf{k}_i^\top]}{\sum_j \exp[\mathbf{q}_1\mathbf{k}_j^\top]}\mathbf{v}_i}_{\in \mathbb{R}^{d_v}} \\ \vdots \\ \underbrace{\sum_{i=1}^{m} \frac{\exp[\mathbf{q}_n\mathbf{k}_i^\top]}{\sum_j \exp[\mathbf{q}_n\mathbf{k}_j^\top]}\mathbf{v}_i}_{\in \mathbb{R}^{d_v}} \end{bmatrix}}_{\in \mathbb{R}^{n \times d_v}} \end{aligned} \tag{59}$$

8.2 Transformer Networks: Scaled Dot-Product Attention

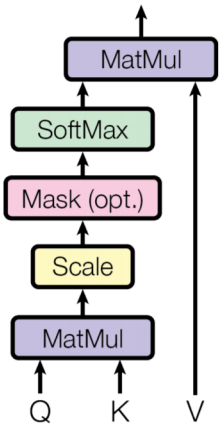
Vaswani, e.t, al, (2017), "Attention Is All You Need" (Google)

the **problem** starts as d_k becomes larger, variance of $\mathbf{q}\mathbf{k}^\top$ increases. hen as a result, some dot product values gets very large, with $\exp(.)$, softmax $\mathbf{p}$ gets peaky! Remember derivative of cross-entropy between softmax **p** and **y** is:

$$\begin{aligned} \mathbf{C}(\mathbf{z}) &= -\sum_{k=1}^{K} y_k \left[\log\left(p_k\right)\right] = -\sum_{k=1}^{K} y_k \left[\log\left(\frac{\exp^{z_k}}{\sum_l \exp^{z_l}}\right)\right] \\ \implies \frac{\mathbf{C}(\mathbf{z})}{\partial \mathbf{z}} &= (\mathbf{p} - \mathbf{y}) \end{aligned} \tag{60}$$

with a peaky softmax, lots of element in gradient vector $\frac{\mathbf{C}(\mathbf{z})}{\partial \mathbf{z}}$ are zero!
the **solution** then becomes that of scaling by length of d_k:

$$A(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \tag{61}$$



"mask (opt.)" is only used at decoder during training

8.3 Self-attention and Multi-head attention

- **generic** input vectors are $(\mathbf{Q}, \mathbf{K}, \mathbf{V})$
- when we let $\mathbf{Q} = \mathbf{K} = \mathbf{V}$, we achieved **self-attention**!
- **even better** can we let words to have multiple ways of interactions with each other?
- **Multi-head attention!**

1. **loop** through $i \in \{1 \dots h\}$, for each i-th iteration:
   - linear transform $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ into several lower dimensional spaces via projection matrices $\mathbf{W}_i^q, \mathbf{K}\mathbf{W}_i^k, \mathbf{V}\mathbf{W}_i^v$ to obtain:

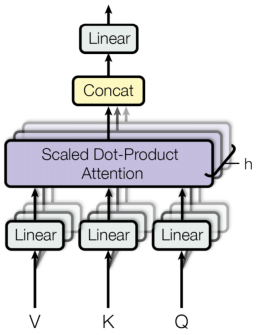
$$\text{head}_i = (\mathbf{Q}\mathbf{W}_i^q, \mathbf{K}\mathbf{W}_i^k, \mathbf{V}\mathbf{W}_i^v) \tag{62}$$

   - each iteration i correspond to one "surface" of the ["linear", "Scaled Dot-Product Attention"] on the diagram
2. then concatenate to produce output matrix $\mathbf{H}$

$$\mathbf{H} = [\text{head}_1, \dots, \text{head}_h] \tag{63}$$

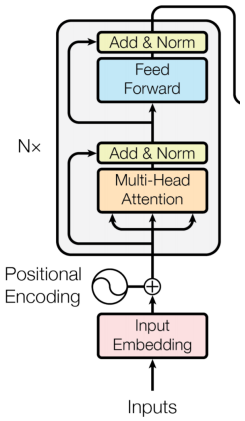
3. finally,

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{H}\mathbf{W}^o \tag{64}$$



8.4 Attention is all you need: encoder

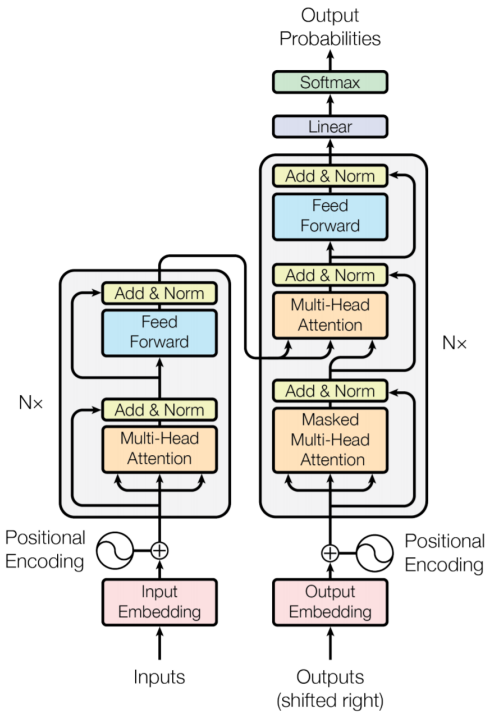
- for **encoder** each block, use the same $(\mathbf{Q} = \mathbf{K} = \mathbf{V})$ from previous layer
- all the goodies from previous deep learning research: ReLU + ResNet+ NN + LayerNorm
- blocks repeated $N\times$ times
- unlike RNN, using attention loose the ordering of the words in encoder, therefore, **explicit position encoding** is required



8.5 Attention is all you need: decoder

- again, all the goodies: ReLU + ResNet+ NN + LayerNorm
- during training: masked decoder self-attention on previously generated outputs
- Encoder-Decoder Attention: $\mathbf{Q}$ come from previous decoder layer and $\mathbf{K}$ and $\mathbf{V}$ come from output of encoder
- the above is similar to **seq2seq with attention** $\mathbf{Q} = \mathbf{h}$ from decoder, $\mathbf{K} = \mathbf{V} = \mathbf{z}$ from encoder
- blocks repeated N times

below we have the figure containing both the encoder and decoder



9 Generative Pre-Training

9.1 Unsupervised pre-training

we attempt to maximize the log-likelihood of the sentence (no markov assumption of course!), maximizing with respect to Θ:

$$\mathcal{L}_1 = \sum_i \log \Pr(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta) \tag{65}$$

let $\mathbf{W}_e$ be token embedding matrix, and $\mathbf{W}_p$ be position embedding matrix, and $\mathbf{U} = (u_{-k}, \ldots, u_{-1})$

$$\begin{aligned} \mathbf{h}_0 &= \mathbf{U}\mathbf{W}_e + \mathbf{W}_p \\ \text{repeat:} \quad \mathbf{h}_l &= \text{transformer\_block}(\mathbf{h}_{l-1}) \quad \forall i \in [1, n] \end{aligned} \tag{66}$$

$\mathbf{h}_n$ is the output of the (last layer) Transformer, and therefore we can compute the word probabilities:

$$\Pr(\mathbf{u}) = \text{softmax}(\mathbf{h}_n \mathbf{W}_e^\top) \quad \forall i \in [1, n] \tag{67}$$

9.2 Supervised fine-tuning

$$P(y | \mathbf{x}_1, \ldots, \mathbf{x}_m) = \text{softmax}(\mathbf{h}_l^m \mathbf{W}_y) \tag{68}$$

$$\mathcal{L}_1 = \sum_{(x,y)} \log \Pr(y|x^1, \ldots, x^m) \tag{69}$$

9.3 combine the two

$$\mathcal{L}_3 = \mathcal{L}_1 + \mathcal{L}_2 \tag{70}$$

References

[1] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig, "Linguistic regularities in continuous space word representations," in *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, 2013, pp. 746–751.

[2] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Marc'Aurelio Ranzato, and Tomas Mikolov, "Devise: A deep visual-semantic embedding model," *Advances in neural information processing systems*, vol. 26, 2013.

[3] Oscar Perez Concha, Richard Yi Da Xu, Zia Moghaddam, and Massimo Piccardi, "Hmm-mio: an enhanced hidden markov model for action recognition," in *CVPR 2011 WORKSHOPS*. IEEE, 2011, pp. 62–69.

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

[5] Minh-Thang Luong, Hieu Pham, and Christopher D Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[6] Ilya Sutskever, Oriol Vinyals, and Quoc V Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[7] Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li, "Modeling coverage for neural machine translation," *arXiv preprint arXiv:1601.04811*, 2016.

[8] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly, "Pointer networks," *Advances in neural information processing systems*, vol. 28, 2015.

[9] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.