Attention in Neural Networks

Erik Ylipää

Research Institutes of Sweden

April 29, 2020

Set of vectors

- Imagine we have some set of vectors we want to make predictions based on
- We'll use the example of natural language in this case. The vectors represents words in a sentence and we would like to predict sentiment of the sentence (does this sentence express a positive or negative opinion).
- For example, the sentences "The movie was really good" or "I head a great time" could be represented by the sets
  - $\{\boldsymbol{x}_{\text{the}}, \boldsymbol{x}_{\text{movie}}, \boldsymbol{x}_{\text{was}}, \boldsymbol{x}_{\text{really}}, \boldsymbol{x}_{\text{good}}\}$
  - $\{\boldsymbol{x}_{\text{i}}, \boldsymbol{x}_{\text{had}}, \boldsymbol{x}_{\text{a}}, \boldsymbol{x}_{\text{great}}, \boldsymbol{x}_{\text{time}}\}$
- Order is not represented, so the following equality holds
  - $\{\boldsymbol{x}_{\text{the}}, \boldsymbol{x}_{\text{movie}}, \boldsymbol{x}_{\text{was}}, \boldsymbol{x}_{\text{really}}, \boldsymbol{x}_{\text{good}}\} = \{\boldsymbol{x}_{\text{movie}}, \boldsymbol{x}_{\text{good}}, \boldsymbol{x}_{\text{was}}, \boldsymbol{x}_{\text{the}}, \boldsymbol{x}_{\text{really}}\}$

Sentiment prediction on word vectors

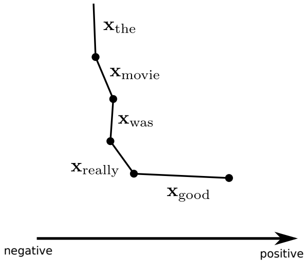
- Now imagine we want to predict sentiment based on the words in a sentence
- We want to define a function from a set of vectors to a sentiment score (e.g. $+1$ for positive sentiment, 0 for negative)
  - $s(\{\boldsymbol{x}_{\text{the}}, \boldsymbol{x}_{\text{movie}}, \boldsymbol{x}_{\text{was}}, \boldsymbol{x}_{\text{really}}, \boldsymbol{x}_{\text{good}}\}) = 1$
  - $s(\{\boldsymbol{x}_{\text{this}}, \boldsymbol{x}_{\text{was}}, \boldsymbol{x}_{\text{the}}, \boldsymbol{x}_{\text{worst}}, \boldsymbol{x}_{\text{movie}}, \boldsymbol{x}_{\text{ever}}\}) = 0$

Sentiment prediction on word vectors using logistic regression

- A straight forward way of doing this is using logistic regression on a fixed length vector representation of the sentence $s = \sigma(W f(\{\boldsymbol{x}_{\text{the}}, \boldsymbol{x}_{\text{movie}}, \boldsymbol{x}_{\text{was}}, \boldsymbol{x}_{\text{really}}, \boldsymbol{x}_{\text{good}}\}) + b), f : \mathbb{X} \rightarrow \mathbb{R}^d$
- The free parameters to learn is then W, b and all the vector word representations $\boldsymbol{x}_{(.)}$.
- We need to define the function f which aggregates the set of words into a fixed length vector (since W and b needs to be of fixed size).
- One of the simplest choices is just vector sum
  - $f(\{\boldsymbol{x}_{\text{the}}, \boldsymbol{x}_{\text{movie}}, \boldsymbol{x}_{\text{was}}, \boldsymbol{x}_{\text{really}}, \boldsymbol{x}_{\text{good}}\}) = \boldsymbol{x}_{\text{the}} + \boldsymbol{x}_{\text{movie}} + \boldsymbol{x}_{\text{was}} + \boldsymbol{x}_{\text{really}} + \boldsymbol{x}_{\text{good}}$

Sums of word vectors for NLP

- We now have a very simple model for sentiment prediction on arbitrarily long sentences:
  - $s = \sigma(W(\boldsymbol{x}_{\text{the}} + \boldsymbol{x}_{\text{movie}} + \boldsymbol{x}_{\text{was}} + \boldsymbol{x}_{\text{really}} + \boldsymbol{x}_{\text{good}}) + b)$
- This model will learn *word polarity*.
  - Words which correlates with positive sentiment will tend to point in one direction in the word vector space
  - words which correlates with negative sentiment will point in the other direction
  - words with little correlation to sentiment will be orthogonal to this direction


$\mathbf{x}_{\text{the}}$
$\mathbf{x}_{\text{movie}}$
$\mathbf{x}_{\text{was}}$
$\mathbf{x}_{\text{really}}$
$\mathbf{x}_{\text{good}}$
negative
positive

Sentiment direction

Issues with sums of word vectors

- Using summation, the sentiment of a sentence is really only determined by sentiment of containing words, not their relation to each other
  - "The movie was really great!" and "The movie was not really great!" will likely have almost the same positive sentiment score.
  - The model can't modify it's prediction based on the presence of "not", since that word vector will not be more strongly correlated with negative sentiment than "great" is with positive
- What if we could weight the vectors depending on context?
  - The presence of "not" could reduce the value of "great"

Weighted sums

- Instead of a regular vector sum, we can instead weight each vector
  - $f(\{\boldsymbol{x}_{\text{the}}, \boldsymbol{x}_{\text{movie}}, \boldsymbol{x}_{\text{was}}, \boldsymbol{x}_{\text{really}}, \boldsymbol{x}_{\text{good}}\}) = a_{the}\boldsymbol{x}_{\text{the}} + a_{movie}\boldsymbol{x}_{\text{movie}} + a_{was}\boldsymbol{x}_{\text{was}} + a_{really}\boldsymbol{x}_{\text{really}} + a_{great}\boldsymbol{x}_{\text{good}}$
  - The a's are scalar and typically constrain the weights so that they are positive and sum to 1, which makes the weighted sum a weighted mean
- For this to make sense we would like the weights to actually be computed from the set at hand
- If they where fixed for each word, it would be like not having them at all since the word vectors are learned)

Weighted view of the sentence

- We define a new function, which has the task of producing the weights for our weighted mean
  - $g(\{\boldsymbol{x}_{\text{the}}, \boldsymbol{x}_{\text{movie}}, \boldsymbol{x}_{\text{was}}, \boldsymbol{x}_{\text{really}}, \boldsymbol{x}_{\text{good}}\}) = \{a_{the}, a_{movie}, a_{was}, a_{really}, a_{great}\}$
- If the weighting function g produces its weights based on the whole context, it's able to change the weight of a word depending on what other words are present in the sentence, e.g. the prescence of the word "not":
  - The movie was really great!
  - The movie was not really great!

Attention in neural networks

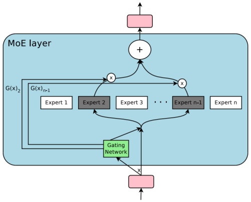


Figure: Mixture of Experts

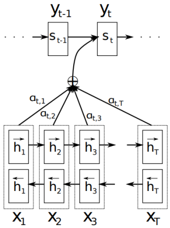


Figure: Information shortcut

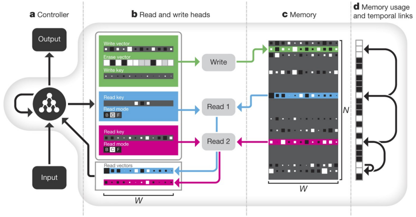


Figure: Memory read and write mechanisms

Shazeer, N. et al. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*

Bahdanau, D. et al. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*

Graves, A. et al. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476

Effect of attention in neural networks

- Attention in neural networks are often explain in terms of what the mechanism could potentially be used for, like in the example of weighted sums for sentiment analysis
- One of the original motivations of attention, as in the mixture of experts examples from the late 1980s, is to isolate parts of the network from change.

Attention computational graph example

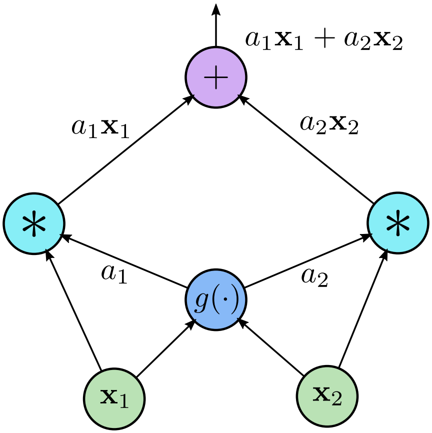


Figure: Simple attention example over two vectors

Attention computational graph example

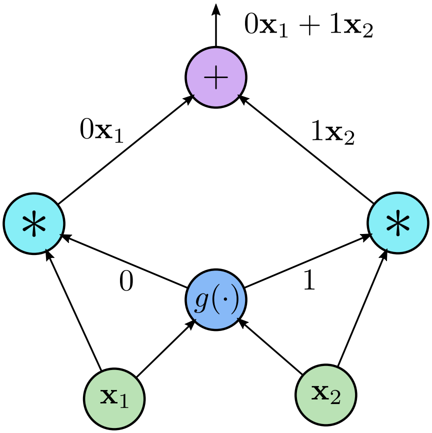
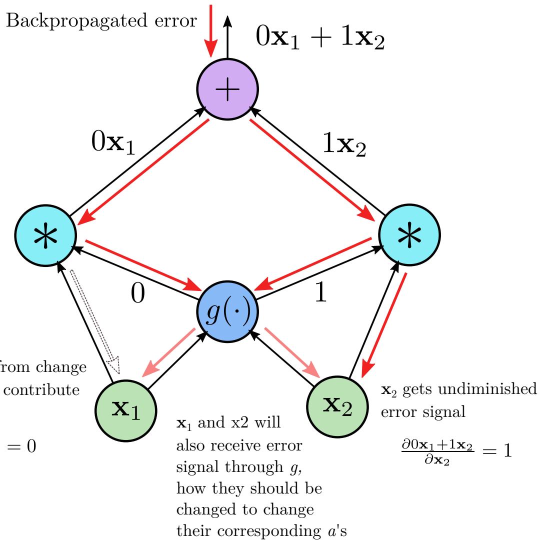


Figure: Example, one vector receives all attention

Attention computational graph example



$\mathbf{x}_1$ is isolated from change since it didn't contribute to the loss

$$\frac{\partial 0\mathbf{x}_1 + 1\mathbf{x}_2}{\partial \mathbf{x}_1} = 0$$

$\mathbf{x}_1$ and x2 will also receive error signal through g, how they should be changed to change their corresponding a's

$\mathbf{x}_2$ gets undiminished error signal

$$\frac{\partial 0\mathbf{x}_1 + 1\mathbf{x}_2}{\partial \mathbf{x}_2} = 1$$

Figure: The attention masks the backpropagated error

Attention variations

- There are many different ways of computing the attention weights, one can divide them into two main strategies:
  1. Fixed attention
  2. Self-attention
- The standard way of making all attention scores be greater than 0 and sum to 1 is to apply the softmax function to them
  - $a_i = softmax(\mathbb{A})_i$, where in our example $\mathbb{A} = \{\alpha_{\text{the}}, \alpha_{\text{movie}}, \alpha_{\text{was}}, \alpha_{\text{really}}, \alpha_{\text{great}}\}$, the α's are the unnormalized attention scores for each element in the set
  - Since the softmax is implicit, we will consider the function g to produce unnormalized scores
- We'll start looking at fixed attention

Fixed attention

- Fixed attention is simple to understand, we learn a function g with parameters $\boldsymbol{\theta}$ which takes a single vector as an input. The unnormalized scores are then
  - $\mathbb{A} = \{g(\boldsymbol{x}_i; \boldsymbol{\theta}) : \boldsymbol{x}_i \in \mathbb{X}\}$
  - e.g. $\mathbb{A} = \{g(\boldsymbol{x}_{\text{the}}; \boldsymbol{\theta}), g(\boldsymbol{x}_{\text{movie}}; \boldsymbol{\theta}), g(\boldsymbol{x}_{\text{was}}; \boldsymbol{\theta}), g(\boldsymbol{x}_{\text{really}}; \boldsymbol{\theta}), g(\boldsymbol{x}_{\text{good}}; \boldsymbol{\theta})\}$
- The function $g(\boldsymbol{x}; \theta)$ could be any function, but in neural networks it has typically been either a neural network with a single hidden layer, or just a dot product with the parameter vector $\boldsymbol{\theta}$.

MLP attention

$g(\boldsymbol{x}; W, \boldsymbol{b}, \boldsymbol{\theta}) = \boldsymbol{\theta}^T \sigma(W\boldsymbol{x} + \boldsymbol{b}), \boldsymbol{x} \in \mathbb{R}^d, W \in \mathbb{R}^{h \times d}, \boldsymbol{b} \in \mathbb{R}^h, \boldsymbol{\theta} \in \mathbb{R}^h$

dot-product attention

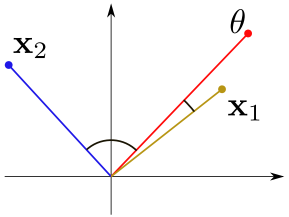
$g(\boldsymbol{x}; \theta) = \boldsymbol{x}^T \boldsymbol{\theta}, \boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{\theta} \in \mathbb{R}^d$

Fixed attention

dot-product attention

$g(\boldsymbol{x};\theta) = \boldsymbol{x}^T\boldsymbol{\theta}, \boldsymbol{x} \in \mathbb{R}^d, \boldsymbol{\theta} \in \mathbb{R}^d$

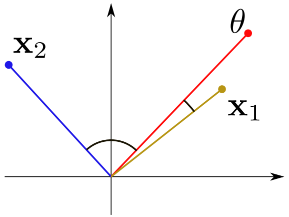
- We'll focus on dot-product attention. MLP attention is very similar, but input vectors are subjected to an affine transformation before the dot-product attention
- I call it fixed, since it will learn a function which gives high score to vectors which have a small angle to its parameter vectors (low angle since both functions use dot products)



The fixed attention function with parameter vector theta will give high scores to input vectors which has a small angle to it

Fixed attention

- This function could help us reweight positive words in the presence of negation
  - assign a very high weight by placing the parameter vector close to negating words, leading to a down-weighting of all other words after normalization, especially if the words we would like to be negated are placed in the opposite direction to the $\boldsymbol{\theta}$ vector
- A single fixed attention function can only learn to be senstive along a single direction in input space, the attention becomes very selective



The fixed attention function with parameter vector theta will give high scores to input vectors which has a small angle to it

Multiple fixed attention functions

- A solution to the issue of fixed attention selectiveness is to have multiple attention functions g
- We can think of them each as being attentive of some particular aspect of the input set
  - $\mathbb{A}_1 = \{g_1(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{X}\}$, $\mathbb{A}_2 = \{g_2(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{X}\}$
  - $a_{i,j} = softmax(\mathbb{A}_i)_j$, the softmax function is applied independently to $\mathbb{A}_1$ and $\mathbb{A}_2$
- In the end, we can concatenate the outputs of the different attentions

$$\boldsymbol{v}_1 = \sum_{\boldsymbol{x}_i \in \mathbb{X}} a_{1,i}\boldsymbol{x}_i, \quad \boldsymbol{v}_2 = \sum_{\boldsymbol{x}_i \in \mathbb{X}} a_{2,i}\boldsymbol{x}_i$$

$$f(\mathbb{X}) = \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \end{bmatrix}$$

Multiple fixed attention functions

- In the case of dot-product attention, this would look (almost) like

$$\boldsymbol{v}_1 = \sum_{\boldsymbol{x}_i \in \mathbb{X}} g_1(\boldsymbol{x}_i)\boldsymbol{x}_i, \quad \boldsymbol{v}_2 = \sum_{\boldsymbol{x}_i \in \mathbb{X}} g_2(\boldsymbol{x}_i)\boldsymbol{x}_i, \quad f(\mathbb{X}) = \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \end{bmatrix}$$

- I say almost since we're omitting the normalization of the outputs of the g's through application of softmax
- We could rewrite this without the g's into

$$\boldsymbol{v}_1 = \sum_{\boldsymbol{x}_i \in \mathbb{X}} ({\boldsymbol{x}_i}^T \boldsymbol{\theta}_1)\boldsymbol{x}_i, \quad \boldsymbol{v}_2 = \sum_{\boldsymbol{x}_i \in \mathbb{X}} ({\boldsymbol{x}_i}^T \boldsymbol{\theta}_2)\boldsymbol{x}_i, \quad f(\mathbb{X}) = \begin{bmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \end{bmatrix}$$

- The difference between g_1 and g_2 is really the parameter vectors $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$

Multiple fixed attention functions

- The difference between g_1 and g_2 is really the parameter vectors $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$
- Instead of considering the attention score functions as being different functions parameterized by $\boldsymbol{\theta}$'s, we can think of it as a single function which takes two arguments:

$$g(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^T \boldsymbol{k}$$

- Where the vector $\boldsymbol{q}$ is the *query*, the vector we wish to calculate attention for and $\boldsymbol{k}$ is the *key*, the vector which determines the attention score.
- In fixed attention, each attention function is now replaced by a different key vector.

General attention framework

- Up until now we have assumed that the vectors we perform the weighted sum on are the same as those used as *queries*. This doesn't have to be the case. A generalized attention function is typically one which takes three arguments

$$f(\mathbb{Q}, \mathbb{K}, \mathbb{V})$$

$\mathbb{Q}$ The query set, each element of this set has a corresponding element in $\mathbb{V}$. We can think of this as how each element of our input set *looks* to the attention function. Another way to think about it is as an indexing set for $\mathbb{V}$.

$\mathbb{K}$ The key set. We can think of this as being *template* vectors the attention function uses to determine what to attend to. Each element of $\mathbb{K}$ will be used to produce one weighted sum of the vectors in $\mathbb{V}$, so the output of our attention will have as many vectors as there are elements of $\mathbb{K}$.

$\mathbb{V}$ The value set. These are the vectors we perform the weighted sum on using the attention scores calculated based on the queries and keys.