Thesis brain dump

Amin Kasrou Aouam

GHENT
UNIVERSITY

10-2021

Contents

Deep Learning **3**
Activation functions . . . 3
Sigmoid . . . 3
ReLu . . . 3

Transformers **4**
Inputs . . . 4
Word representation . . . 4
Positional encoding . . . 5
Self attention . . . 5
Basic mechanism . . . 6
Additional mechanisms . . . 6
Building transformers . . . 7
Example - Text generation transformer . . . 8
Design considerations . . . 9
Modeling of graph data . . . 9

Literature review **9**
CpG Transformer for imputation of single-cell methylomes . . . 9
DNA methylation methodologies . . . 9
CpG site imputation . . . 10
Approach . . . 10

Glossary **13**

References **13**

Deep Learning

Activation functions

Sigmoid

The sigmoid function has a range of [0,1] and can be used both as a **squashing function**, in order to any real number to a value between 0 and 1, or as an **activation** function that guarantees that the output of that unit is between 0 and 1. Furthermore, it is a **non-linear** function and thus ensures that a neural network can learn a non-linearly separable problem.

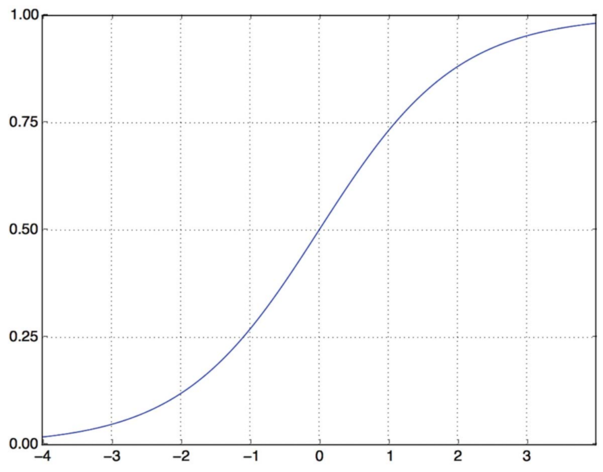

Figure 1: Sigmoid function

A general problem with this function, as an activation function, is that it saturates. This means that large values correspond to 1 and low values to 0, and they are only really sensitive to values around their mid-point. When it is saturated, the learning algorithm has trouble adjusting the weights.

ReLu

The rectified linear unit function or ReLu is a non-linear function that acts like a linear one. Its range is $[0, \infty)$ as it returns 0 for any negative value and the original value if it is positive. In other words, it is linear for positive values and non-linear for negative values.

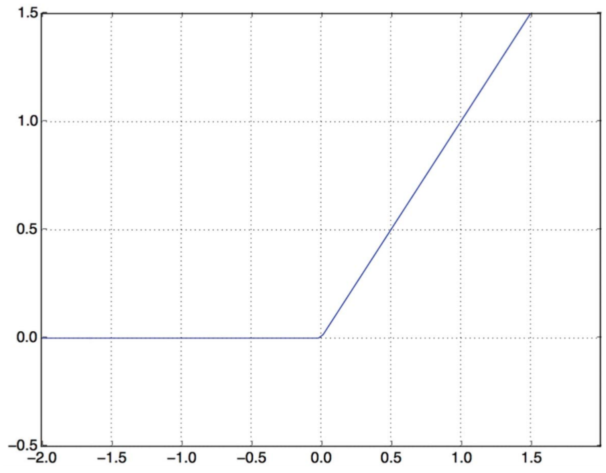

Figure 2: Rectified linear unit function

It is the *de facto* activation function used for Deep Learning networks, due to its many advantages. It is **computationally simple**, allows **sparse representations** (it outputs zero values), has a **linear behavior** (easier to optimize and avoids vanishing gradients).

Transformers

Deep learning models that are design to process a connected set of units (e.g. tokens in a sequence or pixels in an image) only using the mechanism of **self-attention**. They are simple models that still haven't reached their performance limit, as it is currently only being limited by computational resources. They are also **extremely generic**, they have been mostly used for NLP but they can be exploited for more tasks, which is very useful for **multi-modal learning**.

Inputs

Word representation

A word embedding is a featurized representation of a set of words. These high dimensional vectors give a good representation to learn semantic properties of words. A common technique to visualize them is a t-SNE plot, as it plots these high dimensional embeddings into a 2D space. The distance between the points indicates the similarity of the words, which allows us to perform some kind of clustering.

The steps to use word embeddings are the following:

1. Learn them from very large corpuses of unlabeled text/use pretrained word embeddings

2. Transfer embedding to a new task with a smaller training set
3. Finetune the embeddings with new data (optional, useful when the training set is big)

This is a **transfer learning** process.

1. TODO Analogy

   Similarity measures (e.g. cosine similarity)

Positional encoding

Transformers see sentences as sets of words, which means that the order of the words is not relevant. This can be circunvented by using positional encoding, which forces them to evaluate a sentence as a **sequence**. The most common ways of performing it is by using **sine and cosine** functions of different frequencies:

$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$ $PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$

Each dimension corresponds to a sinusoid, which forms a geometric progression from 2π to $10000 \cdot 2\pi$ (Vaswani et al. 2017).

Self attention

It is a sequence-to-sequence operation:

Input vector => Model => output vector

To produce the output vector, the self attention operation performs a weighted average over the input vectors:

$y_i = \Sigma_j w_{ij} x_j$

The weight w_{ij} is not a parameter, but rather it's derived from a function over x_i and x_j. The simplest function is the dot product:

$w'ij = x_i^T x_j$

The softmax function is applied to the dot product in order to map the values to [0,1].

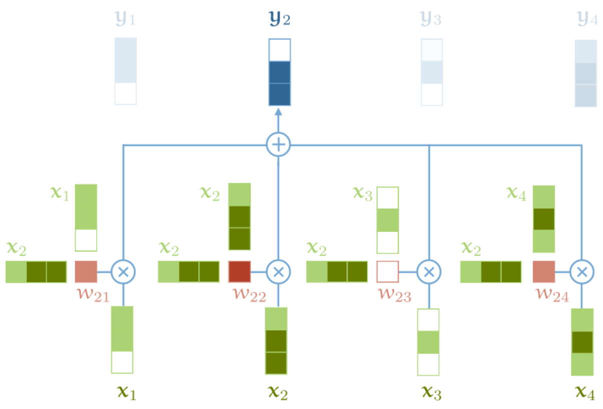

Figure 3: Operation of self attention

The self attention operation is the only one that **propagates information between vectors**.

It is called self attention because there are mechanisms that decide which elements of the input are relevant for a particular output. The general mechanism is as follows, the input are **values**, a mechanism assigns a **key** to each value and to each output the mechanism assigns a **query**. This is similar to how a key-value store works, in our case for each query we will obtain a sum of all the keys weighted by the extent of the match.

Basic mechanism

By using feature selection and performing the dot product, we can apply self attention to NLP. self attention to NLP. By creating an **embedding vector**, which is a numeric representation of a sequence of words we apply the previously formulated y_i function in order to obtain an output vector. The output vector will represent how **related** are two vectors in the input set, in this case related is determined by which learning task we are performing. Self attention sees the **input as set**, the order of the elements is not taken into account.

Additional mechanisms

1. Queries, keys and values

   Each input vector x_i is used in 3 different ways:

   - Query: Compared to every other vector to establish the weights for its own output y_i
   - Key: Compared to every other vector to establish the weights for its own output y_j
   - Value: Used as part of the weighted sum to compute each output vector

In this case we use new vector for each role, which means that we add 3 weight matrices W_q, W_k, W_v and compute 3 linear transformations.

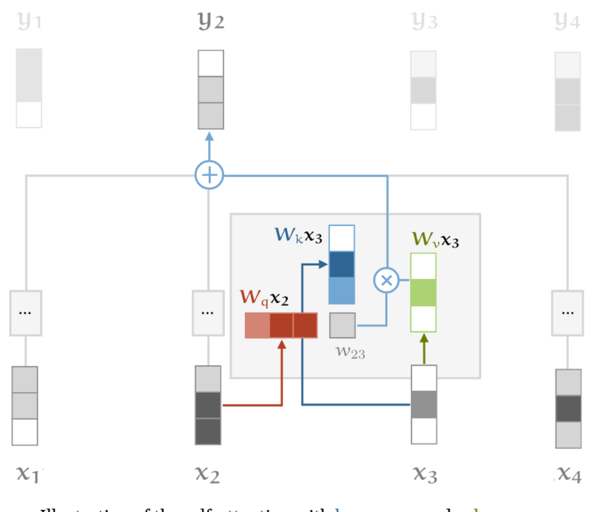

Illustration of the self-attention with key, query and value

Figure 4: Self attention with query, key and value

2. Scaling the dot product

   The softmax function is sensitive to large values, which produce low gradients. We solve this by scaling it down:

   $w'_{ij} = \frac{q_i^T k_j}{\sqrt{k}}$

3. Multi-head attention

   A word can have different meanings depending on its neighbours, in order to work around this problem we combine multiple self attention mechanisms. We assign each attention head a different matrix W_q^r, W_k^r, W_v^r. In order to perform a multi-head self attention efficiently, we divide the input vector by the number of heads ($|x_i| = 256, R = 8$ => 8 chunks of 32 dimensions) and generate queries, keys and values for each chunk.

Building transformers

The standard architecture revolves around 3 types of layers:

- Self attention
- Layer normalization: normalizes the activation of the previous layer **for each sample** in a batch (instead of the whole batch)
- Feed forward layer (MLP)

Residual connections, which allow the neural network to skip them, are added between each layer normalization.

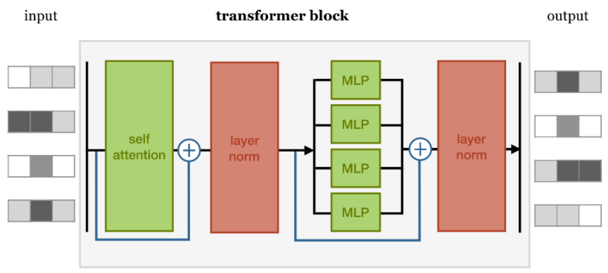

Figure 5: Transformer architecture

The input of the transformer is the embedding vector (word embedding), but in order to take into account the position of the words we need an additional data structure. There are 2 approaches:

- Position embeddings: create an embedding vector containing the position. It's easy to implement but we need to use sequences of every length during the training.
- Position encodings: use a function $f : \mathbb{N} \to \mathbb{R}^k$ to map the positions to vectors of real numbers that the network can interpret. For a well chosen function the network also works on longer sequences, but it is a complicated hyperparameter.

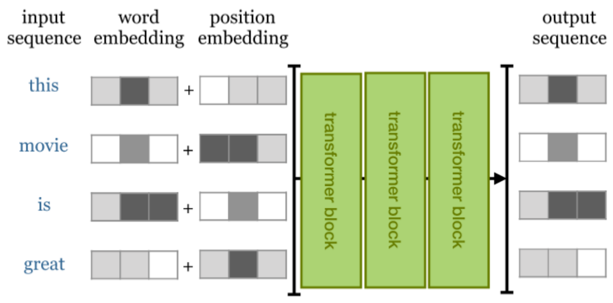

Figure 6: Higher level view of the architecture

Example - Text generation transformer

Transformers can be used as autoregressive models (i.e. they use data from the past to predict the future), one example is a model that predicts the next character in a sequence.

In order to use self-attention for this use case, we need to mask the values after the chosen position i. This is implemented by applying a mask to the matrix of dot products, before the softmax function. The mask sets all the elements above the diagonal to $-\infty$.

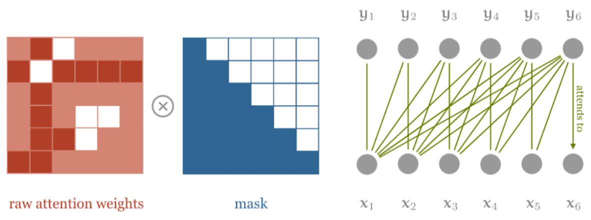

Figure 7: Application of the mask to the dot product

Design considerations

Transformers were created to overcome the shortcomings of RNNs, as the recurrent connection imposes a dependency of the previous timestep to compute the current one.

They can model dependencies over the whole range of the input sequence (unlike CNNs) and they can be computed in a very efficient way. Furthermore, they were designed to allow for deep models, as almost all the model (except softmax and ReLU) are linear transformations which **preserve the gradient**.

Modeling of graph data

Transformers are able to interpret graph data, as they see a sentence as a **fully connected** graph of words. In the case of NLP, it is possible to use full attention as the number of nodes (and subsequently of edges) is small enough which makes it computationally tractable. However, this approach is not possible to interpret most types of graph data such as biological networks. In that case, we need to apply sparse attention (e.g. evaluate the local neighbours).

Literature review

CpG Transformer for imputation of single-cell methylomes

DNA methylation methodologies

DNA methylation is a mechanism that is associated with multiple cellular processes, such as **gene expression**. In the last decade, multiple new single-cell protocols have been developed and although they provide an unprecedented look into cellular processes, they come with some caveats. The smaller amount of reads result in **noisier** data.

CpG site imputation

Prediction of methylation states is a well known problem that has been tackled by leveraging dependencies between sites, using multiple techniques:

- Dimensionality reduction
- Imputation of single CpG sites
- Use of information from multiple tissues
- Use of intra and extracellular correlations
- Differences in local CpG profiles between cells: methylation states at a target sites and its neighbouring ones

Approach

A transfomer model is used to attempt to fill the gaps in a known sequence of methylation states. This is achieved using a mask and then asking the model to predict the masked value. This approach is common in NLP, but has not been explored to **impute gaps in matrices**.

1. Inputs

   - CpG matrix
   - CpG positions in the genome
   - DNA surrounding these sites
   - Cell index embedding -> cell identity

   The CpG matrix is **corrupted** by randomly masking some tokens and 20% of the tokens are also assigned a random binary state.

   Five different datasets were used:

Dataset	Organism	Medium	Platform
20 embryonic stem cells	Mouse	Serum	scBS-seq
12 embryonic stem cells	Mouse	2i	scBS-seq
25 hepatocellular carcinoma cells	Human		scRRBS-seq
30 monoclonal B lymphocytes	Human		scRRBS-seq
122 hematopoietic stem cells	Human		scBS-seq

   Methylation states are assigned when $\frac{\#(reads_{positive})}{\#(reads_{total})} \geq 0.5$ and holdout validation is used (fixed splits).

2. Mechanism

   The model learns a representation for every site and combines them in a graph-like way. It uses **axial** and **sliding** window attention.

   a) Axial attention

      Self-attention is a powerful method but it comes at a high computational cost, as its memory and computation scale quadratically $O(n^2m^2)$, which makes it prohibitely expensive to apply it to long sequences (Ho et al. 2019).

      Axial attention applies attention along **one axis** of the tensor (e.g. height/width of an image) which is faster than applying it on all the elements. It allows for the majority of the context to be embedded, with a high degree of parallelism. This reduces the complexity to $O(mn(n+m))$.

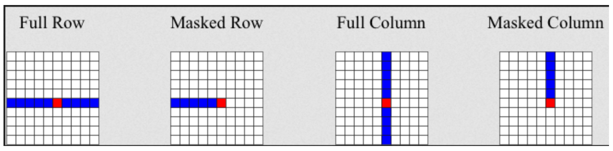

Figure 8: Types of axial attention layers

   b) Sliding window attention

      Sliding window attention employs a fixed-size window of size w around each token, each token then attends to $\frac{w}{2}$ tokens to each side. The complexity of this pattern is $O(n \times w)$, to make this pattern efficient w needs to be smaller than n.

      As CpG sites in close proximity are often correlated, we can apply this mechanism in order to limit row-wise attention, which reduces the complexity to $O(mn(n+w))$.

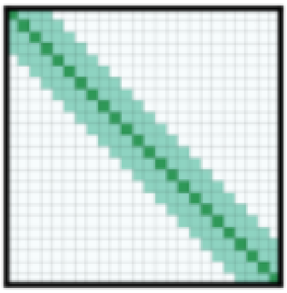

Figure 9: Sliding-window attention

   c) Architecture

The CpG trasnformer is composed of a stack of **four** identical layers, each layer is composed of **three** different sublayers arranged in the following order:

- Sliding-window attention
- Layer normalization
- Axial attention (column wise)
- Layer normalization
- MLP (ReLu activation)
- Layer normalization

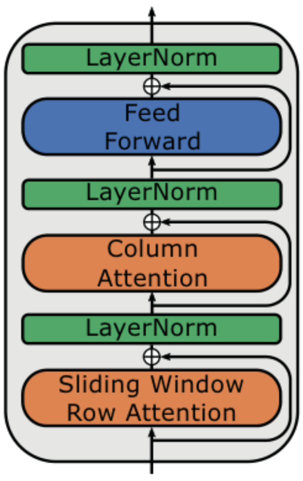

Figure 10: Single layer of the CpG transformer

All sublayers have a **residual connection** and the outputs of the last layer are reduced to **one hidden dimension** and subjected to a sigmoid operation.

3. Objective

   The objective is to impute and denoise the DNA methylation data, and it is based on the masked language model (MLM) which is a a type of denoising autoencoding in which the loss function only acts on the subset of corrupted inputs (Devlin et al. 2019).

4. Results

   It provides a general-purpose way of learning interactions between CpG sites within and between cells. Furthermore, it is also **interpretable** and **enables transfer learning**. It also is evaluated against another DL method DeepCpG and a traditional ML one, CaMelia and it outperforms both of them. Furthermore, the performance gain seems to be more pronounced in contexts with higher cell-to-cell variability, which demonstrates an **ability to encode cell heterogeneity**. Unfortunately, it cannot be scaled to large number of cells. Data subsetting techniques would have to be used in order to apply the model, or alternative attention mechanisms (e.g. clustered attention). Furthermore, its performance is hindered in areas of high sparsity as it is not able to properly estimate local methylation profiles. The local neighbourdhood is important, as in areas with low coverage but with a populated neighbourdhood the results are more accurate.

5. Limitations

Glossary

- Temperature: hyperparameter of neural networks used to control the randomness of predictions, by scaling the logits prior to applying softmax: $\frac{logits}{temperature}$. The higher the temperature, the network is more easily excited and thus results in more diversity and mistakes.
- Ablation: removal of components of the input to evaluate their significance

References

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. "BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding." In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171–86. Minneapolis, Minnesota: Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1423.

Ho, Jonathan, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. 2019. "Axial Attention in Multidimensional Transformers." https://doi.org/10.48550/ARXIV.1912.12180.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. "Attention Is All You Need." https://doi.org/10.48550/ARXIV.1706.03762.