Project Report

DIRECTED VARIATIONAL AUTOENCODER (D-VAE) WITH BERT-STYLE ENCODER DECODER

Abhay Kumar
University of Wisconsin-Madison
Madison, WI, 53715
abhay.kumar@wisc.edu

December 17, 2020

ABSTRACT

This report explores the idea of using BERT style encoder/decoder in a novel DAG variational autoencoder(D-VAE) [1]. To encode DAGs into the latent space, D-VAE model leverage graph neural networks with an asynchronous message passing scheme that allows encoding the computations on DAGs, rather than using existing simultaneous message passing schemes to encode local graph structures. **CODE at** `https://github.com/abhayk1201/799_Project/tree/main/code_notebook`

1 Introduction

Many real-world problems can be posed as optimizing of a directed acyclic graph (DAG) representing some computational task. For example, the architecture of a neural network is a DAG. The problem of searching optimal neural architectures is essentially a DAG optimization task. Discrete nature of DAGs make it a hard problem. However, if we can embed all DAGs to a continuous space and make the space relatively smooth, we might be able to directly use principled black-box optimization algorithms to optimize DAGs in this space, or even use gradient methods if gradients are available.

1.1 Asynchronous message passing scheme

In DAG variational autoencoder(D-VAE), we have an asynchronous message passing scheme to encode the computations on DAGs. The message passing no longer happens at all nodes simultaneously, but respects the computation dependencies (the partial order) among the nodes. For example, suppose node A has two predecessors, B and C, in a DAG. This scheme does not perform feature learning for A until the feature learning on B and C are both finished. Then, the aggregated message from B and C is passed to A to trigger A's feature learning. This feature learning scheme is incorporated in both encoder and decoder in D-VAE.

Variational autoencoder provides a framework to learn both a probabilistic generative model $p_\theta(x|z)$ (the decoder) as well as an approximated posterior distribution $q_\phi(z|x)$ (the encoder).VAE is trained through maximizing the evidence lower bound

$$L(\phi,\theta;x) = E_{z\sim q_\phi(z|x)}[log p_\theta(x|z)] - KL[q_\phi(z|x)||p(z)]$$

The posterior approximation $q_\phi(z|x)$ and the generative model $p_\theta(x|z)$ can in principle take arbitrary parametric forms whose parameters ϕ and θ are output by the encoder and decoder networks. After learning $p_\theta(x|z)$, we can generate new data by decoding latent space vectors z sampled from the prior p(z).

2 D-VAE Encoder

Encoder of D-VAE, which can be seen as a graph neural network (GNN) using an asynchronous message passing scheme.

Similar to standard GNNs, we use an update function $\mathscr{U}$ to compute the hidden state of each node based on its neighbors' incoming message. The hidden state of node v is given by:

$$h_v = \mathscr{U}(x_v, h_v^{in})$$

where, x_v is the one-hot encoding of v's type. (like conv, pool, input, output etc). And, h_v^{in} represents the incoming message to v. h_v^{in} is given by aggregating the hidden states of v's predecessors using an aggregation function $\mathscr{A}$:

$$h_v^{in} = \mathscr{A}(\{h_u : u \to v\}),$$

Compared to the traditional simultaneous message passing, in D-VAE the message passing for a node must wait until all of its predecessors' hidden states have already been computed.

To functions $\mathscr{A}$ and $\mathscr{U}$ have been modeled using neural networks (thanks to the universal approximation theorem) *For our case, we can have x_v as the concatenated representation of entity type and word embedding vector.*

2.1 Implementation

For example, we can let $\mathscr{A}$ be a gated sum:

$$h_v^{in} = \sum_{u \to v} g(h_u) \odot m(h_u)$$

where m is a mapping network and g is a gating network.

To model the injective update function $\mathscr{U}$, we can use a gated recurrent unit (GRU), with h_v^{in} treated as the input hidden state:

$$h_v = GRU_e(x_v, h_v^{in})$$

Here the subscript "e" denotes "encoding". Using a GRU also allows reducing our framework to traditional sequence to sequence modeling frameworks.

D-VAE encoder -

For our case, we can have use BERT or transformer encoder instead of GRU cell. We have to make sure the input sequence to GRU cell, following the Asynchronous message passing scheme, should be passed to the BERT encoder as well. Question- Will bi-directional encoder (as in BERT) will have negative impact on DAG sequence ? I think it should not impact the encoding as the relative ordering of nodes are intact for both forward and backward encoder.

3 D-VAE Decoder

The D-VAE decoder uses the same asynchronous message passing scheme as in the encoder to learn intermediate node and graph states. Similar to encoder, the decoder uses another GRU, denoted by GRU_d, to update node hidden states during the generation. Given the latent vector z to decode, we first use Multilayer perceptron (MLP) to map z to h_0 as the initial hidden state to be fed to GRU_d. Then, the decoder constructs a DAG node by node.

Similarly, we can have use BERT or transformer decoder instead of GRU cell.

4 Few discussions points

1. As opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore it is considered bidirectional,

though it would be more accurate to say that it's non-directional. This characteristic allows the model to learn the context of a word based on all of its surroundings (left and right of the word).

– will this have impact on encoding directed graphs or asynchronous message passing scheme ?

2. **For learning embedding vectors of complex protein/gene names:** Bert works on token level. In addition it learns multi-token embeddings. Lets say we have the word "goodness". Lets say this does not exist in the vocabulary. However, the following tokens exists:
"good"
"ness"
"##ness"
Since this is one word, it will be tokenized as "good"+"##ness". Bert will learn an embedding for ("good"-"##ness") as well as embedding for both "good" and "##ness". It is not as good as if "goodness" existed directly, but reasonable. However: Adding extra words is a bit double edged. If you have a domain specific vocabulary and add "goodness" to one of the empty spots, you will have to train from a random weight. Both "good" and "##ness" have OK embeddings already, and even if it never in the training set have seen "goodness" before, it already have a reasonable embedding to start from. If you training the entire network from scratch, it makes more sense to build a vocabulary that is as efficient as possible, ie require as few tokens as possible.
It could cause problem to learn embedding vectors for complex/compound entity names like "Apolipoprotein E-e4". Also, it seems non-trivial to add more than ~ 100 of new words in BERT vocab list without retraining it from scratch.
(https://github.com/google-research/bert/issues/396)

Cuurrently, BERT breaks into sub-words/phrases that doen't have much semantic meaning as shown in 1. Here is one instance.
sentence-a = "CSF levels of HVA, 5-HIAA, and free MHPG, the major metabolites of dopamine, serotonin, and norepinephrine respectively, were measured in 22 patients with AD."
sentence-b = "These 22 patients were also administered tests of picture-recognition and attentional focusing as part of an earlier experiment."

3. **Using BERT attention head weights as the edge weight brtween two entities:** We plan of using attention weights to decide the edges between entities(nodes). However, BERT doesn't have a single attention weights. It has multi-head attention in each encoder layer.

A specific head captures a specific relationship between the input tokens. Now, if we do that h times (a total of h heads) each encoder block is capturing h different relationships between input tokens.

As in BERT, these attention heads matrices are multiplied with corresponding projections into "Value" subspace and then these concatenated vectors are mapped to embedding dimension.

So, the problem is - how to limit this multi-head attention to just "one". I'm currently evaluating the Possible options are- 1. re-training using only one head at last encoder 2. Keeping only one head out of multi-head based on some metric.

This paper discusses some related ideas if we can use just one attention head- "Are Sixteen Heads Really Better than One?" `https://arxiv.org/pdf/1905.10650.pdf`

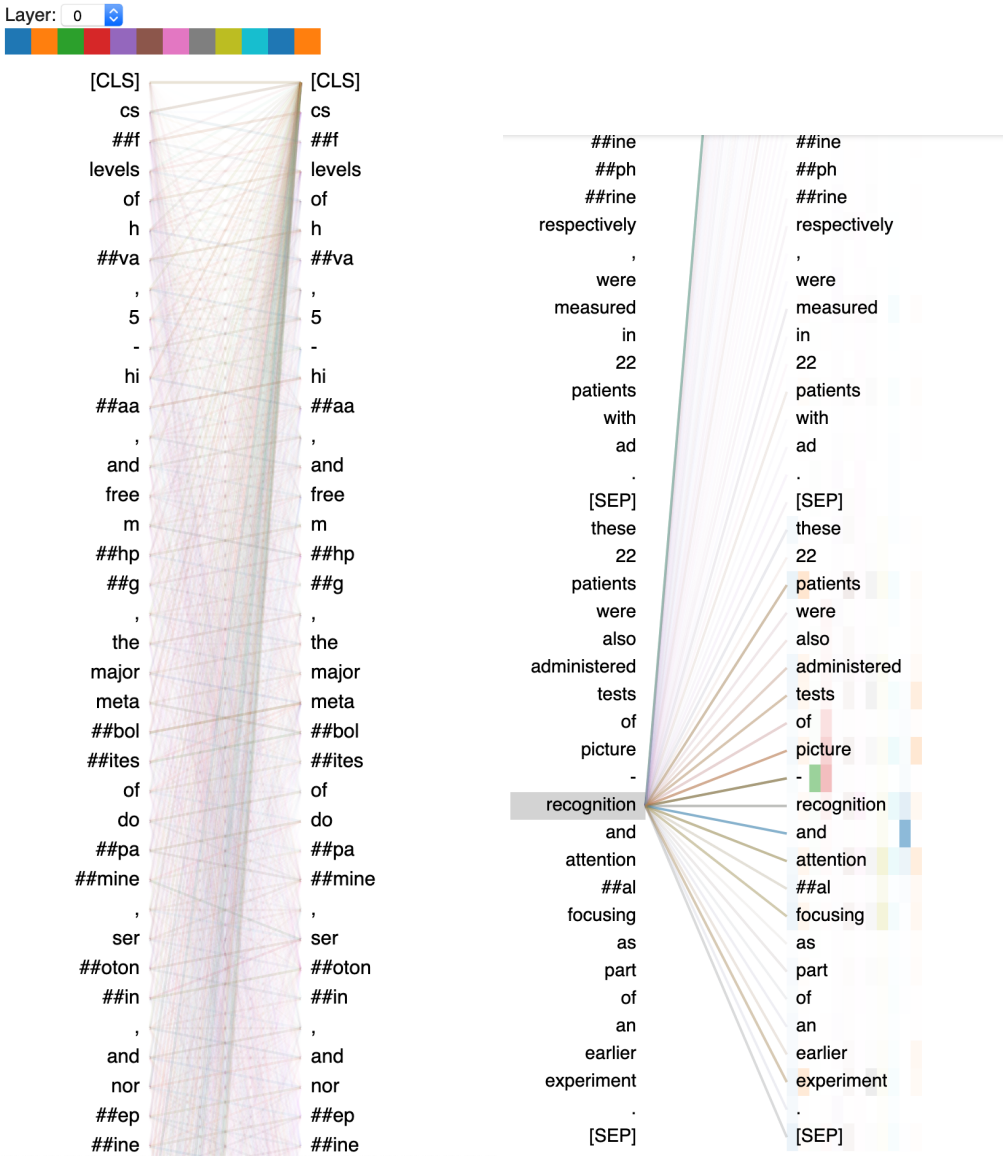

Figure 1: Attention head of BERT. Notice the sub-words don't have much semantic meaning, hence it's unjustifiable to use these weights as the edge weight between two entities.

5 Code

CODE at `https://github.com/abhayk1201/799_Project/tree/main/GNN_PDE`
[1]

References

[1] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: A variational autoencoder for directed acyclic graphs. pages 1586–1598, 2019.

[1] `https://github.com/abhayk1201/799_Project/tree/main/GNN_PDE`