Feature generation using Transformers

Debashish Roy

Why this problem?

Transformers are dominating in many diversified domains of artificial intelligence, be it vision, text, or speech. I have extensively used transformers in NLP domains, and am amazed by their performance. Even just by simple fine-tuning, it can outperform many models. Transformer's last layer mostly gives a unique representation of the input embeddings. These representations can later be used for various purposes. If the representations are very accurate then the problem will boil down to simply making a classifier on N dimension data (where N is the number of features produced by the last layer of the transformer). So learning a good representation is key to many problems. In the case of sequence decision-making, a transformer can capture long-term dependencies and we can use that in our favor to maximize the reward.

Outline

One of the directions I am looking forward to is the “Decision Transformer: Reinforcement Learning via Sequence Modeling” paper. A Decision Transformer is a sequence model that predicts future actions by considering past interactions between an agent and the surrounding environment, and (most importantly) a desired return to be achieved in future interactions. Here State, action, and returns are provided to the time stamp of the transformer, and based on that the model tries to gain maximum rewards based.

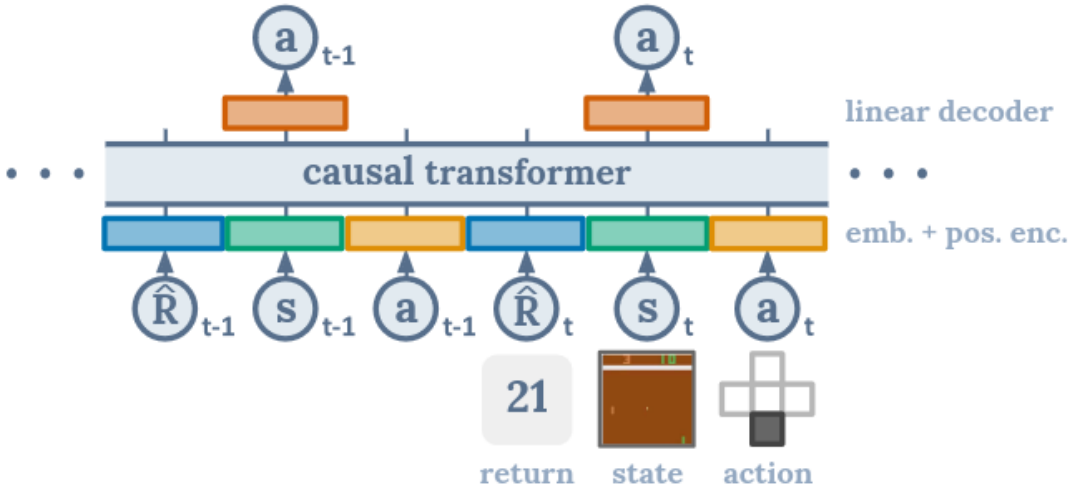

Figure 1: Decision Transformer architecture[1]. States, actions, and returns are fed into modality-specific linear embeddings and a positional episodic timestep encoding is added. Tokens are fed into a GPT architecture which predicts actions autoregressively using a causal self-attention mask.

The idea was to train transformer models on data instead of training for a policy.
Unlike prior approaches to RL that fit value functions or compute policy gradients, Decision Transformer simply outputs the optimal actions by leveraging a causally masked Transformer.

The general idea is as follows:

- We feed the last K timesteps into the Decision Transformer with 3 inputs:
  - Return-to-go
  - State
  - Action
- The tokens are embedded either with a linear layer if the state is a vector or CNN encoder if it's frames.
- The inputs are processed by a transformer model which predicts future actions via autoregressive modeling.

For continuous actions, this can be trained in this way

Algorithm 1 Decision Transformer Pseudocode (for continuous actions)

```
# R, s, a, t: returns-to-go, states, actions, or timesteps
# transformer: transformer with causal masking (GPT)
# embed_s, embed_a, embed_R: linear embedding layers
# embed_t: learned episode positional embedding
# pred_a: linear action prediction layer

# main model
def DecisionTransformer(R, s, a, t):
    # compute embeddings for tokens
    pos_embedding = embed_t(t)  # per-timestep (note: not per-token)
    s_embedding = embed_s(s) + pos_embedding
    a_embedding = embed_a(a) + pos_embedding
    R_embedding = embed_R(R) + pos_embedding

    # interleave tokens as (R_1, s_1, a_1, ..., R_K, s_K)
    input_embeds = stack(R_embedding, s_embedding, a_embedding)

    # use transformer to get hidden states
    hidden_states = transformer(input_embeds=input_embeds)

    # select hidden states for action prediction tokens
    a_hidden = unstack(hidden_states).actions

    # predict action
    return pred_a(a_hidden)

# training loop
for (R, s, a, t) in dataloader:  # dims: (batch_size, K, dim)
    a_preds = DecisionTransformer(R, s, a, t)
    loss = mean((a_preds - a)**2)  # L2 loss for continuous actions
    optimizer.zero_grad(); loss.backward(); optimizer.step()

# evaluation loop
target_return = 1  # for instance, expert-level return
R, s, a, t, done = [target_return], [env.reset()], [], [1], False
while not done:  # autoregressive generation/sampling
    # sample next action
    action = DecisionTransformer(R, s, a, t)[-1]  # for cts actions
    new_s, r, done, _ = env.step(action)

    # append new tokens to sequence
    R = R + [R[-1] - r]  # decrement returns-to-go with reward
    s, a, t = s + [new_s], a + [action], t + [len(R)]
    R, s, a, t = R[-K:], ...  # only keep context length of K
```

We are providing positional embedding to keep track of the token position. This can be used to predict the next state. L2 loss is used for error.

Transformer allows remembering the long-term context which will allow the system to take decisions that will more probably lead to global maximization of reward rather than local optima.

Another aspect is to solve for various input modality data. For example, this is an example of an OSCAR model which can deal with text and image at the same time. It distinguishes between them using a tag variable of image or text. So for problems related to text or vision, we can feature that. I am not proposing that this needs to be used but somewhere in this direction can be explored.

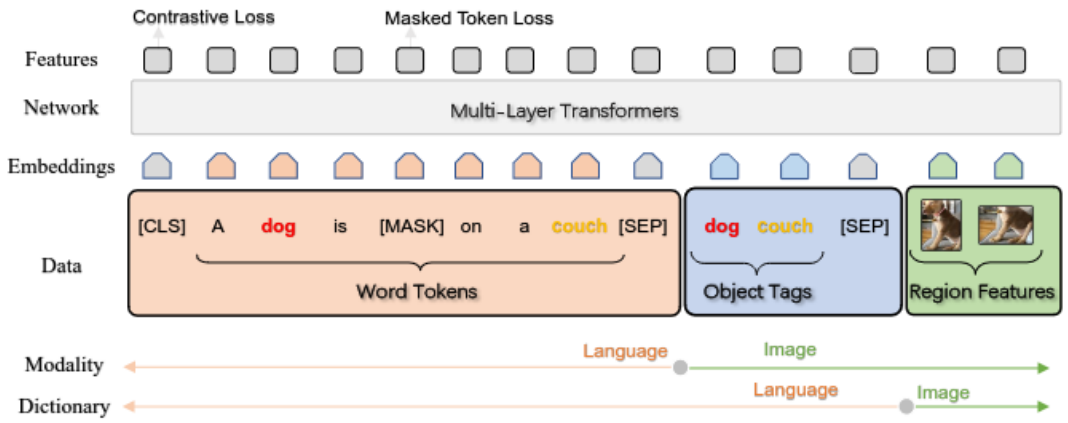

Illustration of Oscar, using text and images simultaneously at one transformer using tags

The same can be extended to audio as well. There are transformers in the audio domain as well, that can help us with various downstream tasks. One of the examples is https://huggingface.co/docs/transformers/tasks/audio_classification.

One of the main advantages of transformers over the RNN family is that they can *parallelize*. This gives an advantage if data is coming in a large volume.

These are some of the cases in which I look forward to working in more detail:

- **Vision Transformer (ViT):** This can be used to capture images which can be a part of the state for example any game (so current position of the game). And later we can devise an autoregressive strategy to get maximum reward.
- **Text-based Transformer:** Often we deal with text. Given a series of interactions, we can use them to predict the maximum rewards with the help of transformers. BERT can be used here.
- **Video-based Representation:** The time series data can be in the form of clips. Here we can use Timesformer to get the clip and predict the next state. I am not sure but this can be used in a generative setup as well.
- **Multi-modal Representation:** OSCAR is an example of this kind of setup.

Code Simulation

I have simulated an experiment, where we are given news of 20 categories (https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html) and we need to classify that.

The code is hosted at https://github.com/debashish05/Reinforcement-Learning-Fest-2023.

- **Transformer.ipynb:** It used SentenceTransformer as a feature generation of 768 size for the text message. Then these features are used to predict the category of the text. This gives an accuracy of 0.0492.

- **Simple_VW.ipynb:** It uses text as an input in raw form. This gives an accuracy of 0.8652. Which is more than using a transformer.

We train both of them in multiclass classification mode, one against all, with hinge loss.

A few things to notice here are:

- The dataset used is 10000, with 768 features it won't formulate well. So here we need to reduce the feature vector size.
- Another thing is the data. Transformer usually works best with a large amount of data. If the data increases, the transformer will improve a lot.
- Nevertheless, we can integrate transformers with VM.
- Transformer takes time to generate features.

Project Timeline (Week-by-Week Plan)

Below is an approximate timeline. If the project finishes early, I will apply the Transformer-generated features to off-policy evaluation.

S.No.	Interval	Task
1.	8th - 20th May	Explore more domains where transformers can be applied. Read the current research paper in this domain.
2.	21st - 28th May	Feasibility Check. Some feature extractors may slow down the system, which needs to be taken care of.
3.	29th May - 10th June	Exploring methods to increase the speed of the algorithm that are giving promising results over current VW solutions.
4.	11th - 25th June	Experiments over the selected algorithm and error analysis
5.	26th - 30th June	Buffer period to catch up on any work that might be left to do.
6.	1st - 7th July	Experiments with online setup at various rates of the input data stream.

7.	8th - 20th July	Implementation and integration of transformer feature with VW.
8.	21st - 25th July	Optimizations to speed up the feature generation process.
9.	26th July - 6th August	Apply the Transformer-generated features to off-policy evaluation
10.	7th - 15th August	Documentation and Testing.

Challenges

- **Data:** Usually transformers need a large amount of data to train on. In case of fewer data, the training won't be appropriate. That is shown in the example of the code where the sentence transformer is not able to perform well due to fewer data points.

- **Time:** The time taken to generate the feature is very high compared to the normal hashing of VW. This can be proved with the code simulation exam that we have taken. One possible approach is to use the Distil version of the models which has less size, but still, we need to customize the transformer based on our requirement to keep the speed of the VW perfect.

Reference

- Oscar: Object-Semantics Aligned Pre-training for Vision-Language Tasks (https://arxiv.org/pdf/2004.06165.pdf)
- Attention is all you Need (https://arxiv.org/pdf/1706.03762.pdf)
- https://mlcourse.ai/book/topic08/topic08_sgd_hashing_vowpal_wabbit.html#news-binary-classification
- https://vowpalwabbit.org/docs/vowpal_wabbit/python/latest/tutorials/index.html
- Decision Transformer (https://arxiv.org/pdf/2106.01345.pdf)
- https://ai.googleblog.com/2022/07/training-generalist-agents-with-multi.html
- https://huggingface.co/blog/decision-transformers