COL775 : Deep Learning

Assignment - 01 Part 2

Text to Math program

Piyush Chauhan

2021CS11010

April 5, 2024

Contents

§1 Overview

In the context of mathematical word problem solving, our task is to automate the conversion of textual narratives into executable mathematical expressions. This assignment entails implementing multiple models to address the challenge. By employing an Encoder-Decoder architecture, particularly seq2seq models, we aim to efficiently translate input text into symbolic form using the operation representation language.

§2 Implementation Details

§2.1 Vocab Building

- **Encoder Vocab**: Encoder vocab was built using tokens from problems present in Training dataset.
- **Decoder Vocab**: Decoder vocab includes tokens from Linear formula present in Traning Dataset.Linear formula was broken at the entity level. For example ``add(n0,n1)|'' was represented as: [``add'', ``('', ``n0'', ``n1'', ``)'', ``|'']
- While building the above two vocabulary , no minimum frequency criteria was kept on a token.
- **Bert-tokenizer**: For bert based encoders, we use bert-tokenizer from transformer library from huggingface. Therefore, we use the corresponding vocab as well.

§2.2 Model 1 : Seq2Seq Model with Glove Embedding

- **Bi-directional LSTM Encoder** was implemented using bi-LSTM cell (`nn.LSTM`) of PyTorch.
- Dropout probability $p = 0.2$ across multiple LSTM cells
- To match the dimentsion of hidden and cell units with decoder input, `torch.cat` and `unsqueeze(0)` was used.
- **Uni-directional LSTM Decoder** Implemented using standard `nn.LSTM` cells with dropout $p = 0.2$
- Pre-trained **GloVe Embeddings** were used from `torchtext` library to initialize the encoder embeddings.
- **Optimizer**: Adam optimizer with 0.001 learning rate, **Scheduler:** CosineAnnealingLR scheduler , **Loss:** Cross entropy loss.

§2.3 Model 2 : Seq2Seq + Attention Model with Glove Embedding

In continuation of the earlier implementation, we have introduced a modification to the decoder component, incorporating an Attention Decoder.
We have developed an **AttentionNetwork** class utilizing `nn.Module` and integrates it with the decoder.Certain adjustments were made to the decoder to incorporate contextual information from the attention network into the input dimension of the decoder LSTM cell.

§2.4 Model 3 :BERT Seq2Seq + Attention Model with frozen BERT-base-uncased

- **Bert Encoder :** We use the BERT base implementation from **huggingface's transformer** library and used the pre-trained weights available by them.
- To freeze the fine-tuning of bert model imported, we use `require_grad = False` for all the parameters of the model.
- We used the `last_hidden_state` of the output as the bert-encoder-output.

§2.5 Model 4 :BERT Seq2Seq + Attention Model with fine-tuned BERT-base-uncased

There are a total of **12-layers** in BERT. For this part, last 3 layers were kept open for find-tuning.

§3 Results

§3.1 Loss Curve Comparison

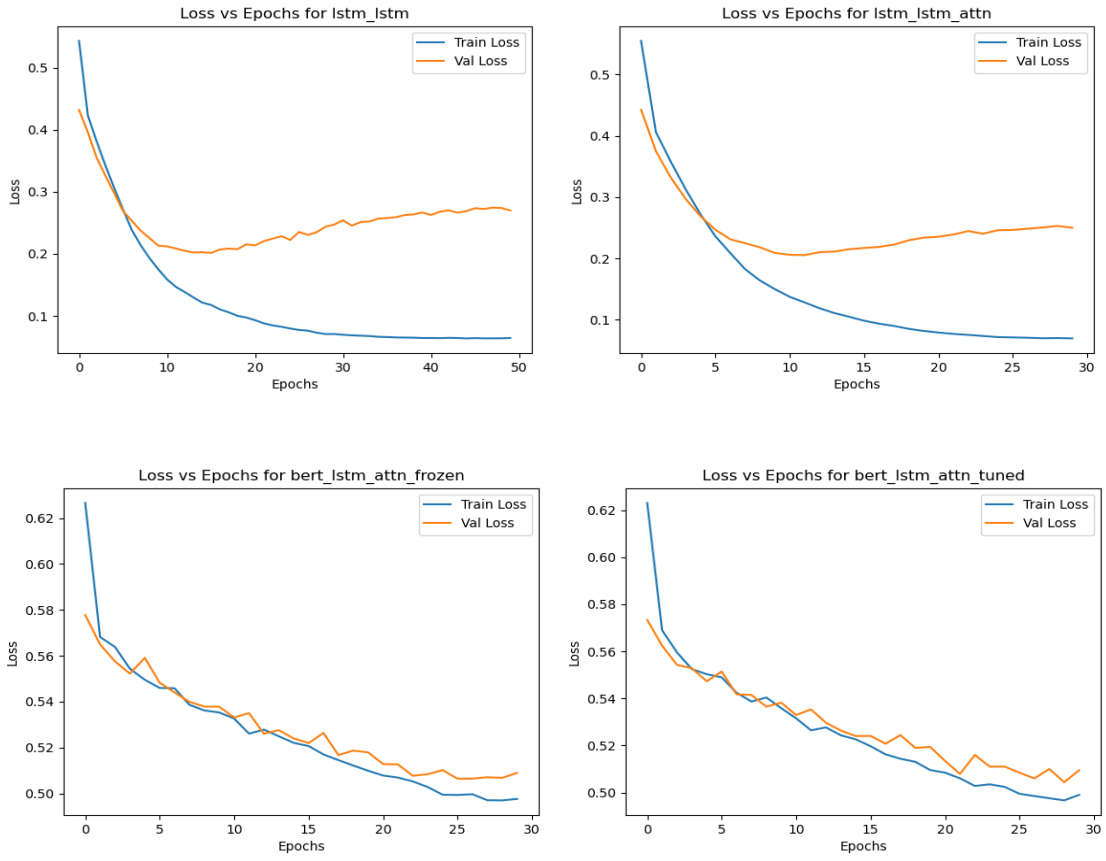

Figure 1: Train and Val loss for models (a) lstm_lstm, (b) lstm_lstm_attn, (c) bert_lstm_attn_frozen, (d) bert_lstm_attn_tuned

Observations from loss curves in Figure 1

- **Train Loss:** For all four curves, train loss reduces continuously with increase in epochs and settles to a low value close to 0.02 for lstm_lstm and lstm_lstm_attn.
- For Bert Based encoders, not much significant loss was observed while training. This can be due to incorrect implemetation of Bert Models. Thus, further claims on Bert models are based on theoretical.
- **Validation Loss:** For lstm models, val loss reduces initially till 10-15 epochs, then it increase a bit and then stays almost smooth towards the end of training. This shows model settles in a local minima valley.

In order to compare the train loss curves and val loss curves with other models, we plot them on two separate graphs in Figure 2.

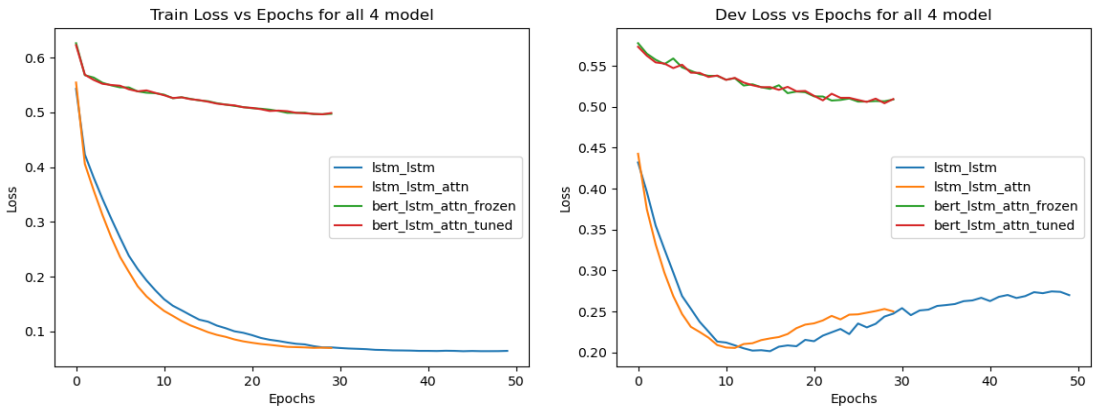

Figure 2: Train and Dev loss curves compared for all the models

Note: As Bert models are not seem to achieve minimum, indicating mistake in implementation, further arguments **do not** include results obtained from bert training.

- **Initial and Final Values**: Initially, both lstm_lstm and lstm_lstm_attn started with 0.55 in Train loss and 0.45 in Dev loss.
- Initial rate of decay is fastest for lstm_lstm_attn in both the curves. Indicating that attention helps in achieving minima early.
- After initial decay, Dev loss curve increases for both the models. However, increase is maximum in lstm_lstm_attn

§3.2 Execution and Exact Match Accuracy for lstm_lstm and lstm_lstm_attn

Dataset	Execution Accuracy(%)	Exact Match Accuracy(%)
lstm_lstm		
Dev Set	64.91	68.57
Test Set	69.53	72.31
lstm_lstm_attn		
Dev Set	62.81	65.97
Test Set	64.87	68.34

Table 1 Train and Dev scores for lstm$_lstmandlstm_lstm_attn$

§3.3 Effect of Teacher Forcing

Below is the train and dev loss graph for TF = 0.3,0.6,0.9

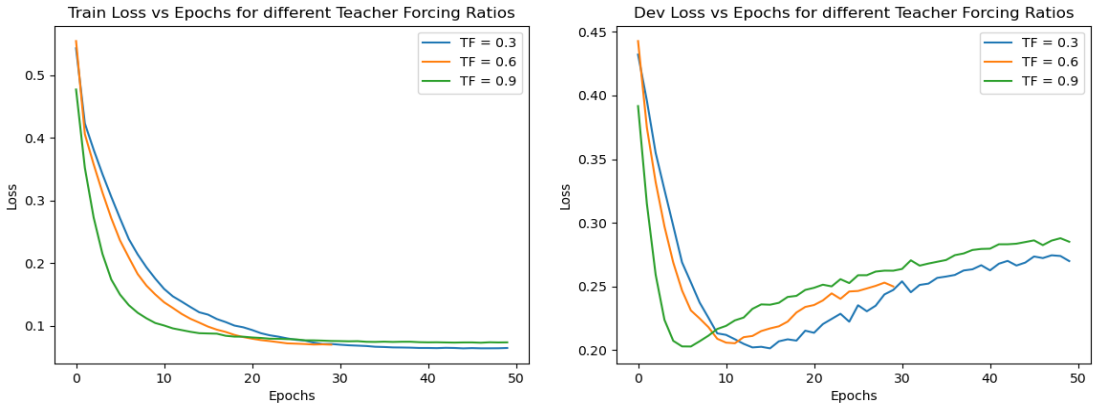

Figure 3: Train and Dev loss curves compared for different TF ration

§3.4 Execution and Exact Match Accuracy for lstm_lstm_attn for different TF ratios

Dataset	Execution Accuracy(%)	Exact Match Accuracy(%)
TF = 0.3		
Dev Set	62.78	65.76
Test Set	64.25	68.13
TF = 0.6		
Dev Set	64.91	68.57
Test Set	69.53	72.31
TF = 0.9		
Dev Set	45.35	48.66
Test Set	49.63	52.54

Table 2 Train and Dev scores for lstm$_lstmandlstm_lstm_attn$

Observations from table and graph

- Modle with higher teacher forcing(TF) is seen to settle at minima faster than other two TF ratios.
- At the end, it is seen that TF =0.3 has minimum Train and Dev loss amongst all.
- Order of convergence $TF = 0.9 \geq TF = 0.6 \geq TF = 0.3$.
- After initial decay, Dev loss curve increases for all variations. however, highest variation is seen model with highest Teacher forcing.
- The drawback of strict/high teacher forcing emerges when the network switches to closed-loop mode/testing time, potentially encountering input disparities between training and testing phases.Thus, an appropriate amount of teacher forcing is observed to perform well in most senarios.

§4 MODELS

MODELS